НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій Стіренко

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Серверна система управління контентом»

Виконав (-ла):
студент (-ка) IV курсу, групи ІО-64
Потапенко Дмитро Олександрович _____

Керівник:
Старший викладач
Саверченко Василь Григорович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Sen. Java Dev., DXC Luxoft, к.т.н.,
Невдащенко М. В. _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій Стіренко
«___»_____2020 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Потапенку Дмитру Олександровичу

1. Тема проєкту «Серверна система управління контентом», керівник проєкту Саверченко Василь Григорович, старший викладач, затверджені наказом по університету від «07» травня 2020 р. № 1081-с
2. Термін подання студентом проєкту *26 травня 2020р.*
3. Вихідні дані до проєкту *див. технічне завдання*
4. Зміст пояснювальної записки *Аналіз і характеристика об'єкта проектування, визначення оптимального варіанта реалізації мети цієї роботи, опис вибраних технологій та етапів розробки програми, візуалізація результату. Висновки.*
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) *принципова схема, функціональна схема, структурна схема*

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Сімоненко В. П., професор, д.т.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми роботи*	*01.09.2019*	
2.	*Вивчення та аналіз завдання*	*15.12.2019-15.03.2020*	
3.	*Розробка архітектури додатку*	*15.03.2020-25.03.2020*	
4.	*Написання програмної частини*	*25.03.2020-05.04.2020*	
5.	*Тестування та виправлення помилок*	*05.04.2020-15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04.2020-20.05.2020*	
7.	*Захист програмного продукту*	*25.04.2020*	
8.	*Передзахист*	*26.05.2020*	
9.	*Захист*	*18.06.2020*	

Студент Дмитро Потапенко

Керівник Василь Саверченко

Анотація

Робота присвячена розробці сервісу, за допомогою якого люди творчих професій зможуть розміщувати свої роботи та резюме. Водночас це все може бути переглянуто іншими користувачами. На даний момент в інтернеті існує велика кількість продуктів, які дозволяють розмістити свої роботи. Моєю ціллю було зробити такий, на якому буде можливість показати свої найкращі проекти та розмістити своє резюме.

Таким чином сервіс стає не лише місцем, де можна виставити портфоліо, але й тим, за допомогою якого можна знайти роботу або професіонала потрібного рівня для найму. Також ідея в себе включає зручність відгуку на вакансії, оскільки роботодавцю достатньо буде відправити лише посилання, в якому він зможе знайти всю необхідну для себе інформацію та відразу ж оцінити можливості та вміння потенційного співробітника.

Аннотация

Работа посвящена разработке сервиса, с помощью которого люди творческих профессий смогут размещать свои работы и резюме. Это всё может быть просмотрено другими пользователями. На данный момент в интернете существует большое количество продуктов, которые позволяют разместить свои работы. Моей целью было сделать такой, на котором будет возможность показать свои лучшие проекты и разместить свое резюме.

Таким образом сервис становится не только местом, где можно выставить портфолио, но и тем, с помощью которого можно найти работу или профессионала нужного уровня для найма. Также идея в себя включает удобство отклика на вакансии, поскольку работодателю достаточно будет отправить только ссылку, в которой он сможет найти всю необходимую для себя информацию и сразу же оценить возможности и умения потенциального сотрудника.

Abstract

The work is dedicated to the development of a service which will help people of creative professions to post their works and resumes. This can also be viewed by other users. At the moment, there are a large number of products on the Internet that allows you to post your works. My goal was to make one where I would have the opportunity to show my best projects and publish my resume.

Thus, the service becomes not only a place where you can put a portfolio, but also one with which you can find a job or a high level professional to hire. The idea also includes the convenience of responding to vacancies, as you will only need to send a link to the employer in which he will be able to find all the information and immediately estimate an abilities and skills of a potential employee.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6421. 02.000 ТЗ	Технічне Завдання	4	
3	А4	ДП 6421. 03.000 ПЗ	Пояснювальна записка	70	
4	А4	ДП 6421. 04.000 Д1	Принципова схема	1	
5	А4	ДП 6421. 05.000 Д2	Функціональна схема	1	
6	А4	ДП 6421. 06.000 Д3	Структурна схема	1	

Зм.		№ документа	Підп.	Дата	ДП.6421.01.000 ВП		
Розробив		Потапенко Д.О.			Серверна система управління контентом Відомість дипломного проєкту	Літ.	Аркуш 1 / 1
Перевір.		Саверченко В. Г.				Т	
Н.конт		Симоненко В.П.				НТУУ «КПІ імені Ігоря Сікорського», ФІОТ Група ІО - 64	
Затв.							

Технічне завдання

до дипломного проєкту

на тему «Серверна система управління контентом»

ЗМІСТ

1.НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: «Серверна система управління контентом».

Область застосування: програма може бути використана будь-яким користувачем для розміщення свого портфоліо та резюме, для пошуку потенційних працівників або просто для поширення своїх особистих даних.

2.ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3.МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення серверної системи управління контентом.

4.ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література та публікації в спеціалізовані публікації в мережі Інтернет по даній темі.

5.ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Програмний продукт, що розробляється повинен:

- Реєструвати нових користувачів сайту та заходити в профіль для тих, що вже зареєстровані;
- Додавати та змінювати персональну інформацію кожного користувача;
- Додавати та змінювати портфоліо кожного користувача;
- Додавати та змінювати резюме кожного користувача;
- Додавати та змінювати оформлення профілю кожного користувача;

5.2. Вимоги для пристроїв

- Мінімальна версія операційної системи для смартфонів - Android 4.0 ;
- Мінімальний обсяг оперативної пам'яті - 1 ГБ;
- Мінімальна частота процесора – 1,1 ГГц;
- Наявність у пристрої підключення до Інтернету;
- Наявність встановленого на пристрої браузера;

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення необхідної літератури	19.02.2019
Складання і узгодження технічного завдання	06.03.2019
Написання вступної частини та огляд рішень	19.03.2019
Розробка структури та визначення можливостей сайту	03.04.2019
Написання програмної частини	09.04.2019
Тестування та виправлення помилок	07.05.2019
Оформлення документації дипломного проекту	16.05.2019
Попередній захист та проходження нормативного контролю	30.05.2019
Захист дипломного проекту	15.06.2019

Пояснювальна записка
до дипломного проєкту
на тему: «Серверна система управління контентом»

Київ – 2020 року

ЗМІСТ

ВСТУП

Сьогодні важко уявити собі життя без Інтернету і без всіх тих зручностей, які він надає. Кожного дня люди користуються цим засобом швидкого поширення інформації і навіть і не замислюються, як все працює, як це організовано і що потрібно зробити для того, щоб воно було у нас вдома за робочим столом чи завжди при собі в телефоні. Це стало на стільки повсякденним, що стає складно уявити свій звичайний день без Гугл-запиту в браузері чи повідомлення в месенджері.

Забезпечення всього цього комфорту можна розділити на велику кількість етапів. Виглядів структури Інтернету є величезна кількість, але візьмемо приблизний або, якщо можна так сказати, «схематичний» опис даного об’єкту. Починаємо з нижнього рівня, котрий назвемо мережею обміну даними, котра включає в себе магістральний канал, потім переходимо до провайдера, котрий, грубо кажучи, підключений до магістралі і надає користувачам свої послуги, далі – клієнти, котрі вже використовують Інтернет кожен в своїх цілях. Але це ще не все. Щоб використовувати цю мережу, потрібен інтерфейс, за допомогою якого і будуть вирішуватися всі ті питання, котрі постають щодня. Ось, який величезний шлях проходить інформація, поки вона дійде від серверів через магістраль та провайдерів і потрапить до нас у кишеню.

Користуються цим функціоналом за допомогою браузерів чи додатків, котрі завантажують сторінки з тим наповненням, яке необхідне. Пошуки здійснюються на абсолютно різні теми і вирішують неймовірну кількість питань, котрі виникають щодня, звичайним запитом в пошуковому рядку. Раніше люди користувалися для цього книгами, газетами, телефонували знайомим – загалом намагалися знайти вихід із ситуації всіма можливими доступними способами. Безумовно, зараз всім цим теж користуються, але, дивлячись правді у вічі, можна впевнено сказати, що в основному це здійснюється за допомогою Інтернету.

Існують спеціалізовані сайти, на котрі заходять, тому що знають, що там є саме те, що треба. Наприклад, на Ютубі шукають відео-контент, на Вікіпедії – текстову інформацію, ну а Гугл, Бінг, Яндекс та інші – пошукові системи, котрі допомагають знаходити абсолютно будь-що в різних напрямках за нашим запитом. Тому, можна зробити висновок, що люди користуються неймовірною кількістю сайтів щодня, аби забезпечити себе комфортом та отримати від Інтернету саме те, що нас цікавить на даний момент.

Актуальність теми

Сьогодні пошук роботи за допомогою Інтернету перейшов на новий рівень і став основним джерелом для даної цілі, саме тому і зросла популярність сервісів, котрі забезпечують такі послуги. Загалом можна знайти купу універсальних, але вони будуть менш зручними, тому що підлаштуватися під всіх неможливо.

Мета і задачі дослідження

Метою роботи є розробка сайту, котрий би був орієнтований на дизайнерів. Проаналізувавши весь «ринок» подібних сервісів, я зрозумів, що немає саме того, який, на мою думку, був би максимально зручним, простим у використанні та мав приємний дизайн. Функціонал повинен включати в себе можливість реєстрації кориістувача, заповнення резюме та додавання прикладів своїх робіт, завантажуючи зображення з готовими результатами. Все поєднається на одній сторінці з можливістю переключатись між вкладкою «резюме» та «проекти». Звісно ж, буде можливість редагувати свій профіль, змінювати та додавати нові роботи і тд.

РОЗДІЛ 1

ОСНОВИ ТА БАЗОВІ ПОНЯТТЯ

1.1 Поява, розвиток, зміна стандартів та постійне вдосконалення

Наприкінці 1960-х військові США розробили комунікаційну мережу під назвою ARPANET. Це можна вважати провісником Інтернету, оскільки він працював над комутацією пакетів і демонстрував першу реалізацію набору протоколів TCP / IP. Ці дві технології складають основу інфраструктури, на якій будується Інтернет. Більш детально з історією можна ознайомитися за посиланням [1]. У 1980 році Тім Бернерс-Лі написав програму для ноутбуків під назвою INQUIRE, в якій була представлена концепція зв'язків між різними вузлами. До 1989 року Тім він написав «Information Management: A Proposal» та «HyperText та CERN»; ці дві публікації разом забезпечили основу для роботи мережі в Інтернеті і стали основою для створення глобальної системи гіпертексту. Наприкінці 1990 року Тім Бернерс-Лі створив усі необхідні для запуску першої версії Інтернет - HTTP, HTML, перший веб-браузер, який отримав назву WorldWideWeb та сервер HTTP. У наступні кілька років, Інтернет вибухнув, було випущено кілька браузерів, створено тисячі веб-серверів та створено мільйони веб-сторінок. В 1994 році Тім Бернерс-Лі заснував World Wide Web Consortium (W3C), організацію, яка об'єднує представників багатьох різних технологічних компаній для спільної роботи над створенням специфікацій веб-технологій. Після цього слідували інші технології, такі як CSS та JavaScript, і Інтернет почав виглядати більше як Інтернет, про який звикли чути сьогодні.

Веб-стандарти. Веб-стандарти - це технології, які використовуються для створення веб-сайтів. Ці стандарти існують як довгі технічні документи, що називаються специфікаціями, в яких детально описано, як саме повинна працювати технологія. Ці документи не дуже корисні для того, щоб навчитися використовувати технології, які вони описують, але натомість призначені для

використання інженерами програмного забезпечення для впровадження цих технологій (як правило, у веб-браузерах). Наприклад,

HTML Living Standard описує, як саме слід реалізувати HTML (усі елементи HTML та їх асоційовані API(Application Programming Interface) та інші навколишні технології). Веб-стандарти створюються органами зі стандартів - установами, які запрошують групи людей з різних технологічних компаній зібратися разом і домовитися про те, як технології повинні працювати найкращим чином, щоб описати всі випадки їх використання. W3C - це найвідоміший орган веб-стандартів, але є й інші, наприклад WHATWG (які відповідали за модернізацію мови HTML), ECMA (які публікують стандарт для ECMAScript, на якому базується JavaScript) та ін.

Відкриті стандарти. Одним з ключових аспектів веб-стандартів, про який Тім Бернерс-Лі та W3C домовилися з самого початку, є те, що Інтернет та веб-технології повинні бути вільними як для участі, так і для використання, а не обтяженими патентами чи ліцензуваннями. Тому кожен може написати код, щоб створити веб-сайт безкоштовно, і кожен може зробити свій внесок у процес створення стандартів, де написані специфікації. Оскільки веб-технології створюються відкрито, у співпраці між багатьма різними компаніями, це означає, що жодна компанія не може їх контролювати, що справді добре. Не хотілося б, щоб колись якась компанія раптом вирішила поставити всю мережу за платною стіною або випустити нову версію HTML, яку кожен повинен придбати, щоб продовжувати робити веб-сайти, або ще гірше, просто вирішивши, що вони більше не зацікавлені і просто відключивши його. Це дозволяє Інтернету залишатися загальнодоступним ресурсом. Одна із основних ідей полягає в тому, що будь-яка нова веб-технологія, яка впроваджується, повинна бути сумісною з тим, що було раніше, тобто старі веб-сайти все ще продовжуватимуть працювати, а майбутні технології в свою чергу будуть сумісні з тим, що є зараз.

1.2 Огляд базових веб-технологій.

Веб-браузери - це програмне забезпечення, яке використовують для перегляду сторінок в Інтернеті.

Протокол передачі гіпертексту або HTTP - це протокол обміну повідомленнями, який дозволяє веб-браузерам спілкуватися з веб-серверами, на яких зберігаються веб-сайти.

HTML, CSS та JavaScript - це основні три технології, які використовуються для створення веб-сайту. Мова розмітки гіпертексту або HTML складається з різних елементів, в які можна загортати (розмічати) вміст, щоб надати йому значення (семантику) та структуру. Простий HTML виглядає так:

```
<span class="text">Sample of plain text </span>
```

Рис. 1.1 – Приклад простого HTML коду.

Каскадні таблиці стилів (CSS) - це мова, заснована на правилах, яка застосовується для стилізації коду HTML, наприклад, встановлення кольорів тексту та фону, додавання меж, анімації або зміна певних інших частин сторінки. Як простий приклад, наступний код перетворить абзац HTML на червоний :

```
.text{
  color: red;
}
```

Рис. 1.2 – Приклад простого CSS коду.

Загалом – це те, що робить візуальний вигляд HTML більш гарним та приємний для сприйняття.

JavaScript - це мова програмування, яка використовується для додавання інтерактивності на веб-сайтах, від динамічного переключення стилів, до отримання оновлень з сервера і до складної 3D-графіки. Нижче наведено приклад JavaScript коду:

```
<script>
    alert( 'JS code example' );
</script>
```

Рис. 1.3 – Приклад простого JavaScript коду.

Крім базових технологій, таких як HTML, CSS та JavaScript, існують різні інструменти, які можна використовувати для полегшення роботи та ефективності, наприклад:

- Інструменти для розробників у сучасних браузерах, які можна використовувати для налагодження коду.
- Інструменти тестування, які можна використовувати для запуску тестів, щоб показати, чи діє код так, як було задумано.
- Бібліотеки та модулі, побудовані на основі JavaScript, котрі дозволяють значно швидше та ефективніше створювати певні типи веб-сайтів.
- Так звані «лінтери», які приймають набір правил і показують місця, де їх не дотримувались.
- «Мініфікатори», які видаляють всі пробіли з файлів коду, щоб зробити їх меншими і тому швидше завантажувати з сервера.

Крім HTML, CSS та JavaScript існує ще один клас мов, який називається мовами зворотної (або серверної) мови, тобто вони запущені на сервері до того, як результат буде відправлений у браузер для відображення. Типовим способом використання серверної мови є вилучення деяких даних із бази та генерування деякого HTML, щоб помістити дані, перш ніж надсилати HTML

до браузера, щоб відобразити його користувачеві. Йдеться про такі рикладні серверні мови, як ASP.NET, Python, PHP та NodeJS та ін.

Найкращі веб-практики. При розробці веб-сторінок основна причина невизначеності пов'язана з вибором комбінації технологій, за допомогою яких треба врахувати максимальну кількість варіантів того, як користувач буде взаємодіяти з сайтом:

- Користувач може переглядати на смартфоні, з невеликим вузьким екраном.
- Користувач може переглядати на ноутбуці Windows із підключеним до нього широкоекранним монітором.
- Користувач може бути сліпим і за допомогою екранного читання взаємодіяти з веб-сторінкою.
- Користувач може використовувати старий комп’ютер, який не може запускати сучасні браузери.

Якщо не знати, які люди користуватимуться, потрібно спроектувати веб-сайт максимально гнучким, щоб усі перераховані вище користувачі могли ним користуватися. Коротше кажучи, треба робити так, щоб Інтернет працював для всіх, на скільки це можливо.

Сумісність між веб-переглядачами - це практика намагатися зробити так, щоб веб-сторінка працює на якомога більшій кількості пристроїв. Сюди входить використання технологій, які підтримують усі браузери, що забезпечує кращу адаптацію для усіх можливих випадків. Це також передбачає велику кількість тестування, щоб перевірити, чи все вдалося отримати в певних браузерах, а потім додаткову роботу над виправленням цих збоїв.

Гнучкий веб-дизайн - це практика зробити функціональність та макети гнучкими, щоб вони могли автоматично адаптуватися до різних браузерів. Очевидний приклад - веб-сайт, який викладений одним способом у широкоекранному браузері на робочому столі, але відображається як більш компактний одноколонний макет у браузерах мобільних телефонів.

Ефективність означає, що веб-сайти завантажуються якомога швидше, і в той же час робить їх інтуїтивно зрозумілими та зручними у використанні.

Доступність означає зробити веб-сайти корисними для якомога більшої кількості людей. Сюди входять люди із вадами зору, слухом, когнітивними вадами або фізичними вадами: молоді чи старі люди, люди різних культур, люди, що користуються мобільними пристроями, або ті, котрі мають ненадійні чи повільні мережеві зв’язки. Інтернаціоналізація означає зробити веб-сайти корисними для людей, які розмовляють на різних мовах. Тут є технічні міркування (наприклад, змінювати макет так, щоб він все працювало нормально для правосторонніх, лівосторонніх або навіть вертикальних мов), і людські (наприклад, використання простої, не-жаргонної мови, щоб не носії мови змогли все зрозуміти).

Конфіденційність та безпека. Ці два поняття пов'язані, але різні. Конфіденційність стосується того, щоб люди могли приватно займатися своєю справою і не були відслідковані. Безпека стосується створення безпечного веб-сайту, щоб зловмисники не могли вкрасти інформацію.

1.3 Конструктори сайтів

З часом бажання створювати сайти з’явилося не лише у професійних розробників та спеціалістів, а й у звичайних людей, які б хотіли власноруч розробити та розмістити свою інтернет сторінку. Саме тому і почали з’являтися конструктори сайтів. Тепер будь-хто може створити веб-сайт. Такі конструктори існують вже багато років, надаючи платформу користувачам будь-якого рівня досвіду для створення веб-сайтів будь-якого типу. Це пов’язано з їх чистими шаблонами, інтерфейсом та розширеним набором інструментів, які підходять практично для будь-якої галузі бізнесу. Деякі плюси включають в себе відсутність навичок в написанні коду чи дизайну, прості у користуванні інтерфейси перетягування та вставляння, безліч функцій, таких як електронна комерція та швидкість запуску. Крім цього, багатьма конструкторами веб-сайтів можна користуватися безкоштовно. Це є

великим плюсом. Запитання в основному зводиться до того, чи варто робити це за допомогою конструктора або, може, краще вибрати інший спосіб. Перед тим, як використовувати будівник веб-сайтів, слід врахувати цілий ряд моментів. Веб-сайт – це перше враження, яке більшість відвідувачів матиме про бренд, сервіс тощо. Відразу треба думати, як він повинен функціонувати, перш ніж приймати рішення щодо розробки сайту в такий спосіб.

1.3.1 Переваги й недоліки сайтів-конструкторів

На даний момент існує багато популярних програмних засобів такого типу. Наприклад, Wix, Squarespace, Weebly, Shopify та GoDaddy. Такі варіанти пропонують чудовий набір інструментів. Розглянемо деякі плюси і мінуси цих популярних автоматизованих розробників.

Якщо планувати використовувати його для свого наступного бізнесу чи проекту, це заощадить гроші. Буде коштувати дешевше, ніж найняти розробника, який займеться сайтом. Новий або молодий бізнес - не єдиний, хто може отримати від цього користь. Доступність розробників веб-сайтів чудово підходить для: ресторанів, нового бізнесу, осіб таких видів зайнятості, як художники, фотографи, музиканти, ремесла тощо. З якістю веб-сайту, який можна отримати за таку низьку вартість, таке рішення, здається, має найбільший сенс. Але не все так просто, як здається. Використовуючи безкоштовний варіант, який пропонують багато веб-конструктори, можна виявити, що кінцевий результат не буде таким, як можна очікувати. Через це є додаткова функціональність, яка може знадобитися. Серед них: вартість доменних імен, особливості електронної комерції, преміум-ліцензія, спеціальні теми або параметри налаштування. Через певні функції високого класу доведеться платити більше, ніж розраховується, за те, щоб отримати веб-сайт в Інтернеті та підтримувати його. Хоча додаткові збори не надто дорогі, важливо подумати про всі функції, які справді потрібні для веб-сайту, і зробити дослідження.

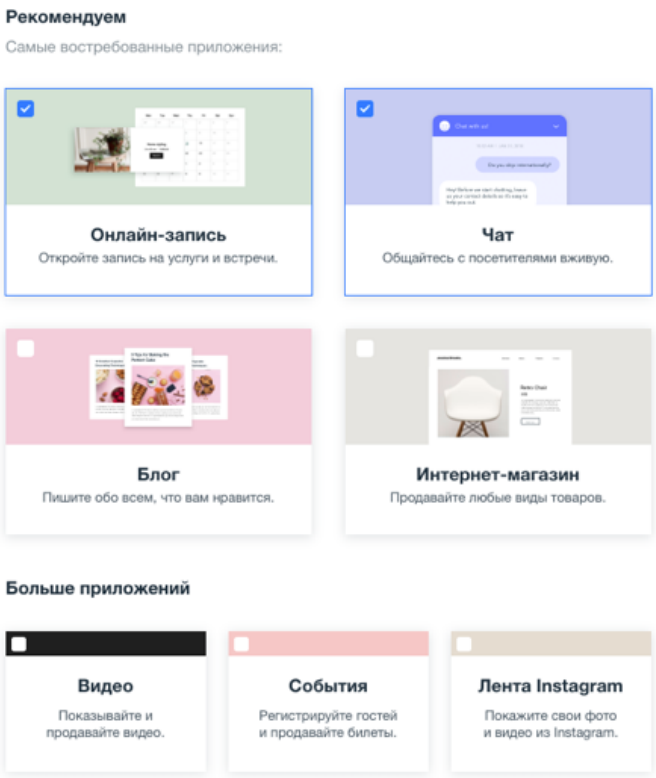


Рис. 1.4 – Приклад функціональності конструктору сайтів [2].

Простота використання. Автоматизовані побудовники веб-сайтів в основному прості. Вони оснащені великими ресурсами, такими як якісні фотографії та відео, які можна використовувати безкоштовно. Багато з них дозволяють створювати інтернет-магазини, прекрасні галереї, круті паралаксальні ефекти, форми електронної пошти для розсилки та інше за допомогою декількох простих клацань. Популярні розробники веб-сайтів, такі як Wix або Squarespace, пропонують прекрасні, сучасні шаблони, які можна вибрати і налаштувати. Тож навіть, якщо раніше не мати досвіду роботи з веб-дизайном, можна створити гарний веб-сайт за рекордно короткий час. Це ідеальне рішення, щоб отримати веб-сайт для своєї перспективної аудиторії в Інтернеті.

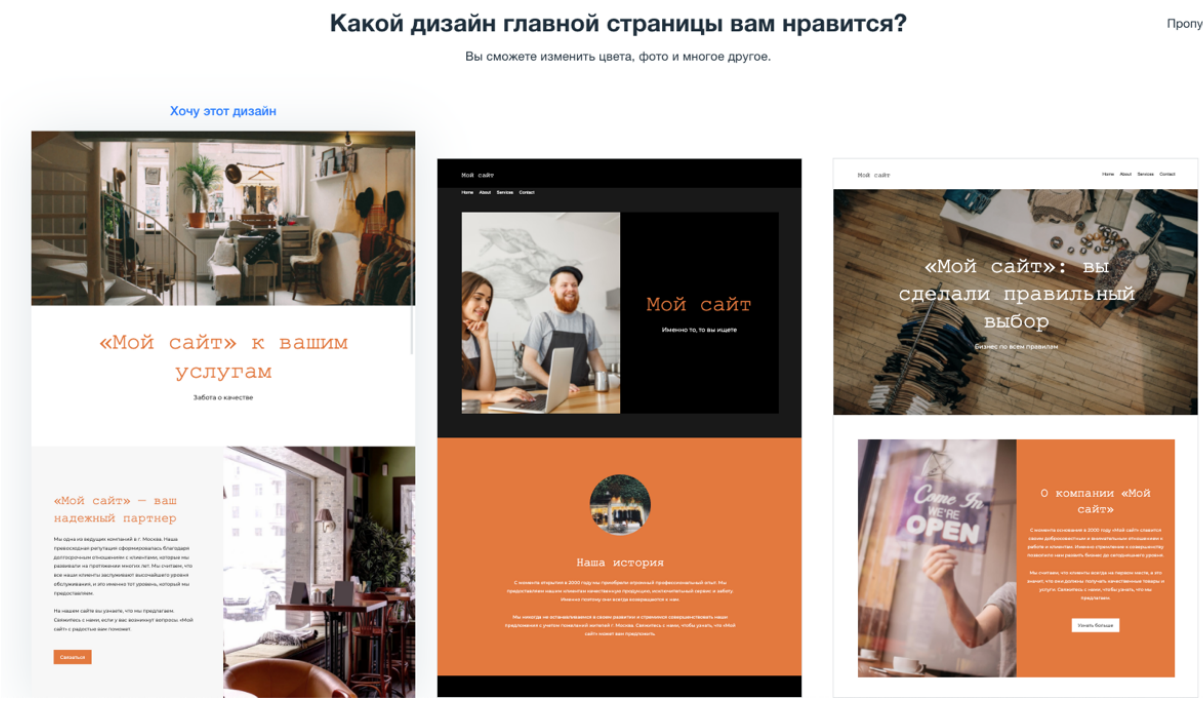


Рис. 1.5 – Приклад дизайну, що надають конструктори сайтів [2].

Все ж таки простота використання сама по собі є суперечкою. Що стосується веб-дизайну, зробити це самостійно простим буде не для всіх. Мало хто знає, що і як має виглядати або як найкраще впровадити дизайнерські чи маркетингові практики. Те, що можна створити чудовий веб-сайт, ще не означає, що він взагалі буде добре працювати. Це все нормально, будівельники допомагають, де вони можуть, і навіть пропонують власну професійну допомогу та підтримку, але знову ж таки, це ще не все. Існують обмеження щодо використання веб-конструкторів сайтів, і вони часто обмежені набором інструментів та меж, які надаються. Через ці обмеження веб-сайт може не з'являтися або не бути дуже унікальним у порівнянні з іншим, побудованим на тій самій платформі, що використовує ту ж тему. Буде важко зобразити сайт унікальним на конструкторі веб-сайту, не маючи навичок чи відчуття дизайну. Також варто пам'ятати, що клієнти та відвідувачі так само бачать якість веб-сайту. Майже завжди помітно, якщо він зроблений за допомогою платформи такого типу і часто може викликати недовіру користувачів. Часто можна бачити надписи на таких сайтах: «Не будьте одним із людей, який відхиляє наш веб-сайт лише тому, що він був створений за допомогою конструктора»

Щодо функціоналу. Можна розраховувати на те, що знайдеться цілий ряд функцій, доступного для кожного розробника. Існують контактні форми, повзунки для зображень, інформаційні поля, інструменти для ведення блогів та багато іншого. Оскільки конструктори веб-сайтів зроблені для різних користувачів, то можна розраховувати на швидку та просту конфігурацію, що відповідає потребам. Можна виділити серед усіх Wix, за те, що він пропонує чудові онлайн-системи замовлення та онлайн бронювання ресторанів і таке інше. Ймовірно, можна навіть знайти інструменти для задоволення більш складної потреби. Але все ж таки є багато обмежень. Налаштувати зовнішній вигляд та функціональність багатьох функцій неможливо. Скажімо, існує форма оренди будинків для відпочинку, і потрібно знати, якщо клієнт привозить домашніх тварин. Скажімо також, що за домашніх тварин системі потрібно стягувати додаткову плату, якщо вибрано цей варіант. Скоріше за все такий функціонал не буде наданий. В кращому випадку він буде наданий лише в преміум-тарифі. Також певні інтеграції або розширені функції можуть мати додаткову вартість. Якщо це так, доведеться платити одноразову або щомісячну плату за цю функцію. Тому треба розглядати необхідні функції та як саме вони повинні працювати до того, як витрачати години, щоб дізнатися про неможливість досягти того, що потрібно.

Рейтинг Google та SEO. Багато розробників веб-сайтів оснащені інструментами оптимізації пошукових систем. Ці інструменти допоможуть класифікуватись у пошукових системах, дозволяючи користувачеві знайти даний сайт в Інтернеті. Незважаючи на те, що кожен такий сервіс мав велику кількість підйомів і падінь, коли мова заходить про SEO, багато з них зробили великі вдосконалення протягом років. Параметри в редакторі допомагають швидко та легко налаштувати веб-сайт, щоб він був зручним для пошукових систем. Інструменти дозволяють змінювати заголовки, мета-описи, вставляти текст та інше, щоб налаштувати, як відображатись у результатах пошуку. Можна навіть відстежувати свої дані про трафік за допомогою вбудованої в Google Analytics. Але незважаючи на вдосконалення, багато веб-сайтів,

побудованих за допомогою конструкторів веб-сайтів, не займають рейтинг популярних пошукових систем, таких як Google. Через обмеження доступу до вихідного коду користувач не може вжити всіх заходів, щоб справді оптимізувати свій сайт. У випадку з конструктором в першу чергу надається контроль над аспектами дизайну веб-сайту, але не над його кодом.

Незалежно від того, підходить автоматизований розробник веб-сайтів для певних цілей чи ні, все залежить від інвестицій: чи буде в результаті те, що хоче клієнт, після того, як на це витратяться кошти.

Підсумувавши, маємо висновок, що немає нічого поганого в тому, щоб скористатися конструктором веб-сайтів, проте завжди краще провести дослідження, перш ніж приймати рішення. Веб-сайт є важливою частиною бізнесу чи бренду. Це відображення обличчя в Інтернеті, і, як результат, одне з перших вражень, яке може скластись у потенційного клієнта. Варто пам’ятати про всі перелічені обмеження і, виходячи із цього, робити висновки: підходить платформа для швидкого збору сайту чи ні.

Висновки до розділу 1

В даному розділі було описано появу Інтернету і швидкий розвиток всіх технологій, котрі забезпечують комфортне його використання. Основними технологіями, котрі є базою у веб розробці є HTML, CSS та JavaScript, котрі теж дуже швидко змінювалися та модернізувалися з поширенням і популяризацією світової мережі. Також було описано альтернативу створення веб сторінок для людей, котрі не мають професійних навичок в цьому напрямку – конструктори сайтів, їх переваги та недоліки.

РОЗДІЛ 2

ОПИС АКТУАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Препроцесори

CSS може бути цікавим, але коли кількість стилів збільшуються, їх стає складніше підтримувати та організовувати. Тут увійшли препроцесори CSS на зразок SASS, Less та інші. Вони надають можливість розробникам frontend розширювати свої навички та можливості за допомогою унікальних функцій, що вдосконалюють CSS. Розглянемо цей питання на прикладі SASS (Syntactically Awesome Style Sheets) - це сценарій мови та препроцесор CSS, який компілюється в CSS, щоб зробити швидші, простіші, більш зручні стилі. SASS додає нові функції та інструменти до базових передбачених можливостей CSS, які допомагають організувати стилізування для великих об’ємів коду. Розширюючи CSS-код, SASS створює все, що є в CSS, і робить великі бази кодів кращими для користувачів. Як відомо, написання CSS може стати досить повторюваним. Препроцесор - це, по суті, сценарій мови, який приймає один тип даних і перетворює його в інший тип. Він розширює можливості за замовчуванням, дозволяючи створювати CSS за допомогою синтаксису мови оригіналу, який можна компілювати в звичний формат. Препроцесор CSS може трансформувати продуктивність та інтерфейс. Він прискорює час розробки та усуває багато обмежень. Розберемо кілька основних переваг використання препроцесорів. Вони генерують чистіший код за рахунок вкладення та змінних. Однією з величезних переваг SASS є можливість вкладати CSS-селектори в селектори. Це означає, що потрібно змінити ім’я елемента лише один раз. Точно так само і з SASS можна використовувати змінні. Це дозволяє зберігати значення та використовувати їх у всіх файлах, спрощуючи загальний код.

За рахунок можливостей повторного використання коду, можна використовувати уже написаний, роблячи всю структуру більш організованою та заощаджуючи час.

```
.log_in_form
    display: flex
    flex-direction: column
    align-items: center
    margin-bottom: 100px
    &-text
        padding: 20px 0
        text-transform: uppercase
        font-size: 20px
        font-weight: bold
    &-block
        width: 400px
```

Рис. 2.1 – Приклад SASS коду.

Існує більше речей, які можна зробити при використанні препроцесору, такі як умови або обчислення. Вони більш функціональні, ніж CSS. Препроцесор додає певний рівень завдяки таким характеристикам, як міксин, вкладення, успадкування, фрагментування тощо. Ці можливості препроцесора полегшують обробку великого обсягу коду. Такі можливості, як змінні та функції, означають, що можна трансформувати CSS-код, зробивши його набагато зрозумілішим та більш організованим. Зрештою, з таким кодом набагато легше працювати, ділитися та редагувати. Таблиці стилів поєднані з HTML та JavaScript для визначення стилів тексту, розміру шрифту тощо для створення рівномірного вигляду на веб-сайті. Зазвичай на сторінці є один CSS документ, який описує стиль для всіх сторінок, на які потрібно посилатися. На жаль, CSS є дещо обмеженим. Ось чому SASS був розроблений як «поліпшений CSS». Він вніс нові функції та легкість. Він має два різні синтаксиси, які часто плутають початківці: SASS та SCSS. Різниця полягає в тому, що SCSS (він же Sassy CSS) - це сучасний стандарт, який використовує дужки та крапки з комою. SCSS був представлений в третій версії SASS як набір CSS. Це викликає стурбованість тим, що SASS використовує зовсім

інший синтаксис. SCSS не розширює стандарт CSS, а просто вдосконалює його синтаксис. Насправді CSS дійсний у синтаксисі SCSS, тому їх легко перетворити між собою. SASS, з іншого боку, є препроцесором CSS і старшим синтаксисом, який використовує відступи для організації та розділення блоків коду. Він по суті забезпечує стислий спосіб написання CSS, що розширює його функціональність.

Щодо функціоналу: змінні забезпечують спосіб зберігання інформації або пакетів, які можна повторно використовувати на аркуші стилів. Наприклад, зберігати значення кольорів або шрифти, які потім можна буде будь-коли повторно використовувати в CSS-коді. Це означає, що якщо треба змінити щось то, все, що потрібно – це оновити лише один раз. Змінні заощаджують тонни часу на переписуванні цього коду, а також запобігають появі помилок на цьому шляху. Змінні особливо корисні для великих проектів. Створити їх досить просто в SASS. Потрібен символ $.

Також SASS додає вкладений синтаксис, що дозволяє організовувати елементи DOM набагато більш чітко. Завдяки цьому не треба переписувати селектори кілька разів, тому внесення змін у таблицю стилів тепер є простим завданням. До речі, читати такий код теж набагато зручніше та приємніше. Міксини дозволяють групувати кілька рядків коду разом, які можна повторно використовувати протягом усього коду. Вони схожі з функціями в інших мовах програмування. Щоб створити міксин, достатньо написати @mixin і додати його до коду, використовуючи директиву @include. Mixin можна оновлювати в будь-який час.

SASS частини дозволяють розбивати файли на менші. Є можливість по суті модулювати CSS, щоб зробити його більш реконструйованим кодом. Можна створити частковий файл, який містить лише відповідний код на розділ. Це корисно, якщо файл SASS стає занадто великим. Назва частки починається з підкреслення _. Потім їх можна імпортувати в будь-який час за допомогою директиви @import. Для того, щоб це все працювало, необхідно в кінці зібрати проект. Саме тут стане в нагоді процес збирання – це

послідовність завдань, які виконуються автоматично після запуску нашого проекту. Усі наші файли генеруються та розгортаються як веб-сервер. Процес збирання може покращити робочий процес, оскільки буде можливість складати, об'єднувати, префіксувати і стискати таблиці стилів за допомогою однієї команди.

2.2 Javascript

Зараз Javascript є одним із найбільш використовуваних мов програмування в напрямку веб розробки. JavaScript виконується на комп’ютерах користувачів, коли вони отримують доступ до сторінки, що означає, що все, що можна зробити в JavaScript, не буде додано в обробку на сервері. Отже, це все робиться на стороні клієнта. Це зробило сайти набагато більш гнучкими для кінцевого користувача та менш дорогими з точки зору серверного трафіку. JavaScript дуже простий у виконанні. Все, що потрібно зробити, - це ввести свій код у документ HTML і сказати браузеру, що це JavaScript.

JavaScript працює на комп’ютерах веб-користувачів, навіть коли вони в режимі офлайн. Він дозволяє створювати інтерфейси з високою гнучкістю, які покращують користувацький досвід та забезпечують динамічну функціональність, не чекаючи реакції сервера та показу іншої сторінки. Він може завантажувати вміст у документ, якщо і коли користувач цього потребує, не завантажуючи всю сторінку - це зазвичай називається Ajax. JavaScript може перевірити, що можливо у браузері, і реагувати відповідно – це називається принципи ненав’язливого JavaScript або іноді – захисного сценарію. Ця мова програмування може допомогти виправити проблеми з браузером або «дірки» в його підтримці - наприклад, проблеми з компонуванням CSS в певних браузерах.

Це дуже достойно для мови, з якої до недавнього часу сміялися програмісти, віддаючи перевагу "вищим мовам програмування". Однією з результатів підняття JavaScript є те, що в наші дні створюються все більш

складні веб-додатки, а для високої інтерактивності потрібні Flash або інші плагіни. Оскільки це веб-стандарт, він підтримується на власних веб-переглядачах (більш-менш - деякі речі відрізняються між браузерами, і ці відмінності обговорюються у відповідних місцях у статтях), і він сумісний з іншими відкритими веб-стандартами.

Використання JavaScript змінювалося з роками його використання. Спочатку взаємодія JavaScript з сайтом зводилася здебільшого до взаємодії з формами, надання відгуку користувачеві та виявлення, коли вони роблять певні речі. Використовувався alert () для того, щоб сповістити користувача про щось, щоб запитати, чи потрібно щось робити, або підказати потрібне поле форми для отримання вводу користувача. Розповідати кінцевому користувачеві про помилки за допомогою оператора alert () – це було все, що можна було зробити в перші дні JavaScript.

Коли браузери почали підтримувати та впроваджувати Модель об'єкта документа (DOM), що дозволяє набагато краще взаємодіяти з веб-сторінками, JavaScript став цікавішим. DOM [3] – це об'єктне представлення документа. Він пояснює і типи, і значення, і ієрархію всього, що міститься в документі. Надається можливість отримувати доступ до будь-якого елемента документа та маніпулювати його зовнішнім виглядом, змістом та атрибутами; створювати нові елементи та застосовувати їх в документі. Це означає, що більше не потрібно покладатися на вікна, рамки, форми та некрасиві сповіщення, і можна відповідати користувачеві в документі в чудово оформленому стилі. Разом з обробкою подій, це дуже потужний арсенал для створення інтерактивних та красивих інтерфейсів. Обробка подій означає, що код реагує на речі, що трапляються в браузері. Це можуть бути речі, які відбуваються автоматично - наприклад, завантаження сторінки - але в більшості випадків реагувати треба на те, що зробив користувач у веб-переглядачі. Користувачі можуть змінити розмір вікна, прокрутити сторінку, натиснути певні клавіші або клікнути на посилання / кнопки / елементи за допомогою миші. Під час обробки подій, можна чекати, коли ці речі

відбудуться, і повідомити веб-сторінці, щоб відповісти на ці події. Якщо в минулому натискання будь-якого посилання призвело б до переходу на іншу сторінку то тепер, є можливість захопити цю функціональність і зробити щось інше: як показ і приховування панелі, так і перенесення інформації у посилання та використання її для підключення до веб-сервісу.І це в основному те, що робиться в ці дні з JavaScript.

Також покращується старий, перевірений і справжній веб-інтерфейс - натискання посилань, введення інформації та надсилання форм тощо, щоб бути більш гнучкими для кінцевого користувача. Наприклад, форма реєстрації може перевірити, чи присутнє ім’я користувача, коли його вводять, з відсутністю перезавантажування сторінки. Інформація, яка постійно змінюється, може періодично завантажуватися без необхідності взаємодії з користувачем. JavaScript може вирішити проблеми з компонуванням. Використовуючи його, легко знайти положення, область будь-якого елемента на сторінці та розміри вікна браузера. Скажімо, наприклад, є меню з кількома рівнями; перевіривши, чи є вільний простір для появи підменю перед його відображенням, можна запобігти прокрутці або перекриттю елементів меню. JavaScript може покращити інтерфейси, які надає HTML. Хоча приємно мати поле для введення тексту, можливо, необхідно зробити комбіноване поле, яке дозволяє вибрати зі списку заданих значень або ввести своє. Використовуючи JavaScript, стає можливим покращення звичайного поля введення.

Ця мова широко використовується для анімації елементів на сторінці - наприклад, для показу та приховування інформації або виділення конкретних розділів сторінки - це може зробити більш корисним та багатшим користувацький простір. Слід пам’ятати, що JavaScript може бути недоступним - це легко перевірити, тому насправді дана ситуація не є проблемою. Однак речі, які залежать від JavaScript, слід створювати з огляду на це, і треба бути обережними, щоб сайт не порушувався, тобто щоб основна функціональність була доступною, якщо JavaScript не відповідає. «Якщо використання JavaScript не допомагає користувачеві досягти мети швидше та

ефективніше, ймовірно, він використовується його неправильно» – одне із основних умовних правил. З JavaScript, часто порушуються умови, до яких люди звикли протягом багатьох років користування Інтернетом (наприклад, натискаючи посилання, щоб перейти на інші сторінки). Хоча ці схеми використання можуть бути застарілими та неефективними, їх зміна все ще означає змусити користувачів змінити спосіб - і це змушує людей відчувати себе не зовсім зручно. Нові рішення JavaScript повинні проявляти себе краще, ніж попередні, але й бути не настільки відрізняючими, щоб користувач не зміг зрозуміти, як цим користуватися, використовуючи свій попередній досвід.

JavaScript ніколи не повинен бути заходом безпеки. Якщо потрібно заборонити користувачам отримувати доступ до даних1, то не варто покладатися на JavaScript. Будь-який захист JavaScript може бути легко подоланий, оскільки весь код доступний для читання на клієнтській машині. Також користувачі можуть просто вимкнути JavaScript у своїх браузерах.

Зробимо невеликий підсумок: JavaScript - це чудова технологія для використання в Інтернеті. Він простий у вивчення і дуже універсальний, чудово взаємодіє з іншими веб-технологіями, такими як HTML та CSS - і навіть може взаємодіяти з плагінами, такими як Flash. JavaScript дозволяє створювати інтерпретації користувачів, які дуже гнучко реагують, запобігають перебоям у перезавантаженні сторінок і навіть виправляють проблеми з підтримкою CSS. Використовуючи потрібні додатки веб-переглядача його можна навіть використовувати для того, щоб зробити доступними онлайн-системи в режимі офлайн та автоматично синхронізуватись, коли комп'ютер перейде в Інтернет.

2.3 Найпопулярніші бібліотеки та модулі Javascript

Внаслідок стрімкого розвитку інтернет індустрії розвиваються і засоби за допомогою яких весь потрібний функціонал буде реалізований. Через це і з’являється велика кількість надлаштувань які спрощують, пришвидшують та оптимізують роботу.

2.3.1 AngularJS.

Починається все в 2009 році, коли Мішко Гевери (тепер старший тренер з питань комп'ютерних наук і Agile в Google) та Адам Абронс (нині директор з інженерії в Grand Rounds) працювали над своїм побічним проектом, поступовим інструментом веб-розробки. що запропонувало б послугу зберігання в Інтернеті JSON, а також бібліотеку на стороні клієнта для створення веб-додатків залежно від нього.

Для публікації свого проекту вони взяли ім'я хоста GetAngular.com. Перший стабільний реліз AngularJS (версія 0.9.0, також відомий як драконове дихання) був випущений на GitHub у жовтні 2010 року за ліцензією MIT; коли AngularJS 1.0.0 вийшов у червні 2012 року, рамки вже досягли величезної популярності у спільнотах веб-розробників у всьому світі.

AngularJS була першою базою для клієнта, яка його застосувала: це, безперечно, величезна перевага перед конкурентами, включаючи бібліотеки, що управляють DOM, такі як JQuery. За допомогою AngularJS розробники могли записувати складно зв'язані компоненти, залишаючи в рамках завдання їх створення, вирішення їх залежностей та передачі їх іншим компонентам за запитом. Директиви можуть бути описані як маркери на конкретних елементах DOM, таких як елементи, атрибути, стилі тощо. Потужна функція, яка може бути використана для визначення спеціальних та багаторазових HTML-подібних елементів та атрибутів, що визначають прив'язку даних та іншу специфічну поведінку компонентів.

Автоматична синхронізація даних між компонентами моделі та представлення: коли дані в моделі змінюються, подання відображає зміну; коли дані в представленні даних змінюються, модель також оновлюється. Це відбувається негайно та автоматично. Завдяки чому модель та вигляд постійно оновлюються. Підхід з однією сторінкою AngularJS був першою основою, яка повністю усунула необхідність перезавантаження сторінок. Це забезпечило великі переваги як на сервері (менше мережевих запитів), так і на рівні клієнта (плавніші переходи, більш приємна візуалізація) та проклало шлях для

шаблону додатків для однієї сторінки, який також буде перейнятий React та Vue. Всі дії AngularJS повинні були відбуватися на стороні клієнта без будь-яких зусиль на стороні сервера для створення частин UI / UX. Саме з цієї причини всі веб-сайти AngularJS можна кешувати в будь-якому місці та робити доступними через CDN (мережу доставки вмісту). Інформацію про більш детальний розвиток та поступове вдосконалення цієї бібліотеки можна знайти тут [4].

Оскільки Angular використовується найменше серед всіх найпопулярніших, то його синтаксис описаним не буде.

2.3.2 React.js

За останні кілька років кількість завантажень пакетів React.js з NPM порівняно з іншими двома популярними бібліотеками – Angular, Core та Vue – значно прискорилася. Він забезпечує фантастичну роботу користувачів та розробників. Більш того, його популярність зростає, оскільки її підтримують Facebook та активне співтовариство. Загалом він являє собою інструмент для побудови інтерфейсів. Деякі програмні інженери вважають за краще не називати це "фреймворком", оскільки це дає розробникам набагато більше свободи, ніж попередньо згадані Angular або Vue. Обидва є відкритим кодом та можуть безкоштовно користуватися ліцензією MIT. React.js лише сім років, що робить його відносно новою технологією. Однак вона розвивається дуже швидко. За допомогою JavaScript можна створювати динамічні програми, де браузер виконує значну частину функцій, тому вони можуть працювати, не звертаючись до сервера. Він також дозволяє самостійно оновлювати дані та інтерфейс лише в частині програми без перезавантаження.

Проаналізувавши все те, що було сказано в [5], можна сказати, що React - це інструмент для створення як компонентів інтерфейсу, так і цілих інтерфейсів користувача - все, що стосується об'єднання візуальних елементів, прив’язки даних до цих елементів та визначення логіки, що ним керує. React.js

можна використовувати для створення інтерфейсів користувача у JavaScript для різних платформ, а також використовувати ReactDOM для веб-додатків,

Такоє існує React Native для розробки мобільних додатків (Android та iOS) та міжплатформенних гібридних програмам з Electron. Нещодавно Microsoft також випустила React Native для Windows.

React.js - це технологія інтерфейсу, але вона також може бути виконана на бекенді та використана для десктопних додатків. Існує два можливі підходи до використання сучасних фреймворків JavaScript - візуалізація на стороні клієнта, де браузер завантажує код і надає користувальницький інтерфейс, або візуалізація на стороні сервера, де інтерфейс відображається на бекенді. Різниця між рішеннями JavaScript (такими як React.js) і старими технологіями полягає в тому, що JS переймає набагато більше логіки та маніпулювання документами, так, як ніби це взагалі не було виведено сервером. Основна особливість React.js, яка відрізняє його від інших популярних фреймворків JavaScript, - це гнучкість. Можна взяти бібліотеку і використовувати її для відображення простої сторінки або перегляду, але також можна комбінувати React.js з іншими інструментами і використовувати його як додаток, який закладе основу для складної програми. Це більше, ніж просто бібліотека, оскільки має яскраву екосистему, що складається з інструментів, надлаштувань та підходів. Вони можуть також використовувати власні фреймворки та набір інструментів.

Різниця між React.js та React Native. Нещодавно React Native став ще популярнішим. Це тому, що Facebook активно просуває його як найкращий інструмент для розвитку платформних мобільних додатків. React Native використовує елементи інтерфейсу користувача, написані на React.js, які можуть генерувати вбудовані компоненти інтерфейсу iOS та Android, такі як кнопки та анімації.

React Native дає змогу створювати додатки, які безперебійно працюють на смартфонах iPhone, Samsung чи Huawei і ділять переважну більшість кодів між двома платформами. Ключових терміни, які потрібно знати про React.js:

- Компоненти - це ті будівельні блоки, які можна скласти для створення програми. З React.js досить легко створювати спеціальні компоненти, що є дуже важливою особливістю, оскільки це потрібно в 99% випадків, а звичайні компоненти в основному складають 10-20% компонентів у програмі React.js .Можна також скористатися наявною повноцінною бібліотекою інтерфейсу користувача (наприклад, Material UI) та просто з'єднати компоненти за допомогою даних та власної логіки.
- JSX - це розширення JavaScript, що дозволяє розробникам використовувати синтаксис, що нагадує HTML та XML, який не можна змішувати з JS для управління логікою.

Redux - це бібліотека управління станом з яскравою екосистемою, яка часто поєднується з React.js. У 2012 році Facebook Ads став складним для управління, оскільки веб-додаток соціальної мережі ставав більшим і включав все більше і більше компонентів.

Марк Цукерберг заявив, що покладатися на HTML5 - одна з найбільших помилок їхньої організації, пообіцявши водночас користувачам (та інвесторам), що незабаром Facebook покаже чудовий мобільний досвід. У травні 2013 року офіційно запустили React.js. З його допомогою виробляються нові продукти, а деякі найбільші та найуспішніші включають React.js у свій стек. Сюди входять найвідоміші світові програми, такі як:

- Соціальні мережі (Facebook, Instagram, Pinterest, Twitter)
- Спільний доступ до економічних платформ (Airbnb, Lyft, Uber)
- Медіа-сайти (Yahoo!)
- Відеоплатформи (Netflix)

Facebook створив React для власних цілей; більшість їх веб мобільних додатків написані за допомогою React. На даний момент соціальна мережа перебуває в процесі відновлення всієї платформи за допомогою React.

Ось п’ять найбільш вражаючих функцій React.js, обраних командою Netguru:

- Декларативність. Будується користувальницький інтерфейс програми, створюючи перегляд за переглядом. За допомогою React.js код фокусується на тому, що відображається, а не на тому, які кроки потрібно вжити для його відображення. Для людини далекої від цього, другий підхід може здатися більш інтуїтивним. І справді швидше створити та налагодити екран або компонент цим методом.
- Компонентний підхід. Створюється програми на React.js за допомогою будівельних блоків - компонентів, які можуть відповідати за функцію інтерфейсу користувача (наприклад, кнопки), мережевого зв'язку (наприклад, вибору даних) або набагато складніших функцій, таких як управління станом (наприклад, Redux). Цей модульний підхід дозволяє легко запровадити систему дизайну та показати її.
- Мінімалізм. React.js невеликий і швидкий для завантаження. Для налаштування середовища програмування не потрібно багато роботи. Більше того, не завжди потрібно завантажувати всю програму. Завдяки функції розбиття коду можна значно скоротити час завантаження веб-сайту чи веб-програми. Плавний та швидкий час завантаження мають вирішальне значення для UX продукту, а також для маркетингу, оскільки Google просуває сайти з коротким часом завантаження.
- Величезна екосистема та гнучкість. Вибираючи технологію для цифрового продукту, хочеться використовувати мову програмування або бібліотеки, які вже популярні, і найголовніше, які залишаться популярними в майбутньому. Широке використання надає найважливіші ресурси – спілку талановитих розробників, які знають технологію та готові вивчати та вдосконалювати її, а також розширюють бібліотеки компонентів та нових областей застосування. У React.js все це є.
- Зворотна сумісність. Програмне забезпечення завжди чудово підходить для роботи зі старими версіями бібліотек, які він використовує (або залежить від них). Сьогодні не дуже багато мов та фреймворків можуть

забезпечити сумісність із зворотним рівнем, оскільки це дорого. На щастя, Facebook довелося інвестувати у підтримку зворотної сумісності для внутрішнього корпоративного використання. Деякі додатки FB все ще виконують фрагменти коду, створені, коли React ще був у зародковому стані і саме тому із цим немає проблем.

Даний фреймворк допомагає швидко створювати веб-додатки та гібридні мобільні програми. Це ідеально підходить для сучасного підходу до розробки програмного забезпечення. За допомогою React.js можна запустити MVP дуже швидко, не приносячи шкоди жодному зовнішньому вигляду і UX.

Завдяки своїй модульності React.js легко масштабується. Можна завантажувати продукт з мінімальними ресурсами, а також розробити об'ємну програму, таку як, Facebook. Завдяки наявності та різноманітності бібліотек інструментів та супутніх надлаштувань, що підходять для різних випадків використання, можна використовувати React для створення програмного забезпечення від невеликих, легких веб-сайтів, порталів та додатків середнього розміру, до великих корпоративних систем. React.js ідеально підходить:

- коли треба розробити додатком середньої та високої складності, що вимагає складних потоків на стороні клієнта.
- коли треба створити додаток, що вимагає реактивного та ефективного інтерфейсу користувача.
- при розробці статичного веб-сайту з більш-менш динамічним вмістом.

Насправді, складніше називати сценарії, де React.js не є правильним вибором. Наприклад, якщо треба створити просту програму для створення, читання, оновлення та видалення (CRUD), React.js може бути зайвим. Те саме може бути справедливим у випадку простої разової веб-сторінки. Однак якщо функції є лише першим кроком для подальшого розвитку, то можна розглянути цю технологію. Це працює сьогодні, а завтра буде працювати ще краще. Багато компаній переходять на React.js або починають нові проекти, використовуючи його. Facebook та інші користувачі React.js інвестують у

збереження її масштабованості та безпеки. Ці цілі порівняно легко досягти завдяки мінімалізму React.js. Чим більше компаній та приватних осіб використовують React.js, тим він стає більш гнучким, масштабованим та безпечним, оскільки вся громада працює разом над вдосконаленням технології. Він стає доступнішим, універсальнішим і простішим у використанні.

React.js - гнучкий інструмент, який вже має багато випадків використання. Він може бути реалізований практично в будь-якому проекті, який вимагає Інтернету і має хоча б якийсь рівень складності. Бібліотеку можна використовувати в стартапах для швидкого випуску MVP та згодом будування повноцінного продукту.

Як було сказано вище Javascript має декілька фреймворків, котрі допомагають з вирішенням питанням декомпонування проекту. Найбільш популярними на даний момент є React, Angular та Vue. Оскільки саме React зараз розвивається швидше за усіх, було вирішено використовувати саме його і будувати сайт з використанням наданого ним функціоналу та можливостей. Декомпонування – розділення проекту на декілька частин, з котрих буде складатися проект. В свою чергу ці частини можуть бути використані не лише один раз, що робить структуру досить адаптивною, зручною та функціональною. Візьмемо за приклад звичайно форму реєстрації. Вона складається з декількох подібних полів. Її буде досить зручно побудувати за допомогою компонентів React, оскільки кожне поле являється одним і тим же самим компонентом. Тому його можна створити лише один раз і використати стільки разів, скільки полів буде потрібно. Поговоримо більш детально про те як повинен будуватися проект за допомогою обраного фреймворку. Є 3 основні правила, котрих дуже бажано дотримуватися:

- Все, що повертає компонент, повинно бути загорнута в div
- Компонент повинен вирішувати не лише одну задачу і повинен робити це якісно
- Створювати новий файл для кожного компоненту

```
ReactDOM.render(
  <Car name="Ford" year="2016" />,
  document.querySelector('#root1')
)
```

Рис. 2.2 – Приклад створення React компоненту.

Всередині компонентів потрібно використовувати не звичний html код, а Jsx код, котрий читається реактом. Для початку роботи з React потрібно створити html з тегом div і присвоїти йому id = “root“– це і буде основний розділ в котрому буде розміщуватися додаток React.

```
import React from 'react';
import ReactDOM from 'react-dom';

function Hi() {
  return <div>Hello World!</div>;
}

ReactDOM.render(<Hi/>, document.querySelector('#root'));
```

Рис. 2.3 – Приклад коду з використанням React.

JSX не є рядком. Також React не перетворює ці речі безпосередньо в рядки. Перед запуском React є додатковий крок, що проходить через код, який перетворює JSX у виклики функцій. Hello World! стає React.cre-ateElement('div', null, 'Hello World!'). Останній рядок - це те, що змушує його працювати. Він доручає React викликати функцію «Hi», отримує повернений JSX та вставляє відповідні елементи HTML у документ під елементом з id "root". document.querySelector("#root")працює аналогічно jQuery $("root"), знаходить і повертає цей елемент з документа.

```
ReactDOM.render(<Hi/>, document.querySelector('#root'));
```

Рис. 2.4 – Візуалізація React.

Коли React малює компонент, він передає параметри компонента як перший аргумент, об'єктом. Об'єкт параметру - це просто звичайний JS – об’єкт.

```
function Hi(props) {
  return <div>Hello {props.name}!</div>;
}
```

Рис. 2.5 – Передача props.

Передача параметру дуже схожа на встановлення атрибутів на тег HTML. Багато синтаксису JSX запозичено з HTML. Також можна передавати їм все, що завгодно, наприклад: булеві значення, цифри, рядки, функції та навіть об’єкти.

Всередині компонента, який отримує кілька параметрів, кожен з них буде окремою властивістю переданого об’єкта «props». Наприклад, якщо компонент під назвою «HiFullName», взяв два параметри, то всередині компонент буде виглядати приблизно так:

```
<HiFullName firstName="Dave" lastName="Ceddia" />
```

Рис. 2.6 – Передача 2х параметрів у props.

```
function HiFullName(props) {
  return (
    <div>Hi {props.firstName} {props.lastName}!</div>
  );
}
```

Рис. 2.7 – Показ внутрішніх компонентів.

Можна також часто побачити useState, імпортований з React. Після цього можна звернутися в useState безпосередньо. Те, як використовується, React.useState може виглядати трохи дивно. React здатний запам’ятати стан проміжних викликів до компонента. Перш ніж React викликає компонент, він налаштовує масив для відстеження того, що викликається. Під час виклику useState, React використовує цей масив, щоб відслідковувати його початкове значення, яке змінюється з часом.

Візуалізація на основі state. Зараз компонент працює так само, як і раніше, тому не змінилося нічого у фактичному JSX, який виводиться. Це відображатиметься по-різному, залежно від стану світла.

```
<div className="room">
  the room is {isLit ? 'lit' : 'dark'}
</div>
```

Рис. 2.8 – Візуалізація зі state.

Як можна побачити, світло все ще true. Тепер React.useState(true) переходить до React.useState(false). Додаток повторно відображатиметься, сказавши, що «в кімнаті темно». Це доводить можливість змінювати вивід зміною коду.

```
<div className="room">
  the room is {isLit ? "lit" : "dark"}
  <br />
  <button onClick={() => setLit(!isLit)}>
    flip
  </button>
</div>
```

Рис. 2.9 – Зміна стану по кліку.

Після натискання кнопки спрацює onClick. Щоб імпортувати стилі треба написати import "./index.css"; Тепер потрібно динамічно змінити на room lit або room dark залежно від стану. Після створення стану у верхній частині компонента створено змінну для освітленості і далі її використовуємо таким чином:

```
<div className={`room ${brightness}`}>
```

Рис. 2.10 – Виклик змінної.

```
function Reddit() {
  const [posts, setPosts] = React.useState([]);

  return (
    <div>
      <h1>/r/reactjs</h1>
      <ul>
        {posts.map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
```

Рис. 2.11 – Приклад відображення списку.

Всередині <ul> вираз JS загорнутий в фігурні дужки (це потрібно робити в JSX). Posts – це порожній масив публікацій, який ініціалізований вище, а map функція перебирає повідомлення за повідомленнями та повертає значення <li> для кожного елемента масиву. Далі кожен елемент масиву перетворюється на <li> елемент JSX з key та заголовком повідомлення, і отриманий масив <li> - це те, що отримується всередині <ul>. Внаслідок таких перетворень можна розгорнути дану функцію в такому вигляді:

```
return (
  <div>
    <h1>/r/reactjs</h1>
    <ul>
      {[
        <li key={1}>Post one</li>,
        <li key={2}>Post two</li>,
        <li key={3}>Post three</li>
      ]}
    </ul>
  </div>
);
```

Рис. 2.12 – Зміна стану по кліку.

Для даного прикладу потрібно буде імпортувати axios: import axios from 'axios';

```
React.useEffect(() => {
  axios.get(`https://www.reddit.com/r/reactjs.json`)
    .then(res => {
      const newPosts = res.data.data.children
        .map(obj => obj.data);
      setPosts(newPosts);
    });
}, []);
```

Рис. 2.13 – Використання useEffect.

Тут використовується новий useEffect. Це працює таким чином: передається функція і useEffect піднімається після цього вгору, щоб ця функція була запущена після завершення рендеру. Цей особливий ефект вимагає axios.get отримати дані з API Reddit, який повертає проміс, і .then обробник отримає виклик, як тільки все буде завершено. API Reddit повертає публікації в досить глибоко вкладеній структурі, тому res.data.data.children.map... вибирає окремі повідомлення з вкладень. Нарешті, він оновлює posts стан, викликаючи setPosts.

2.3.3 Vue.js

Головна відмінність Vue від інших монолітних каркасів полягає в тому, що Vue цілком прийнятний, починаючи з його створення. Оскільки основний шар орієнтований лише на шар перегляду, Vue легко інтегрується в проекти. Це означає, що його не потрібно довго налаштувувати. Більше того, Vue дуже добре орієнтований на створення потужних односторінкових додатків.

Синтаксис шаблону Vue.js заснований на HTML, що дозволяє нам пов'язувати DOM рендерінгу з первинними даними екземпляра Vue. Оскільки всі шаблони Vue.js засновані на HTML, їх можна легко проаналізувати за допомогою специфічних браузерів та HTML-парсерів. З іншого боку, Vue також використовує концепцію Virtual DOM.

Virtual DOM - це просто об’єкт JavaScript, який представляє модель об'єкта документа (DOM). Програма оновить Virtual DOM і не доведеться безпосередньо оновлювати «вручну». Більшість бібліотек JavaScript оновлює DOM, впроваджуючи зміни у Virtual Dom.

Vue збирає шаблони у функції візуалізації віртуального DOM. Використовуючи систему реактивності, Vue здатний автоматично визначати мінімальну кількість компонентів, які підлягають рендерингу, і зменшувати маніпуляції з DOM, коли стан додатка змінюється. Для написання додатку Vue треба створити новий екземпляр. Зазвичай, екземпляр називають Vue як vm:

```
var vm = new Vue ({
  // параметри
})
```

Рис. 2.14 – Створення екземпляру.

Шаблон дизайну Vue був створений на основі MVVM (Model-View-ViewModel – шаблон проектування архітектури додатку), але він не зовсім такий. Під час створення екземпляру Vue, слід перейти в об’єкт, щоб створити бажану поведінку. Властивості всередині data об’єкта є реактивними, це означає, що коли вони змінюються, View шукає оновлення та відновлює DOM за допомогою "реагування" на зміни.

```
var vm = new Vue ({
  data: {
    name: 'Ilkin',
  }
})
```

Рис. 2.15 – Зміна об’єкту data.

Компоненти є багаторазовими частинами для застосування. Ця концепція широко використовується у Vue.js та інших фреймворках, таких як

React та Angular. У будь-якій програмі не хочеться, щоб усі дані, методи, обчислені властивості тощо знаходились у кореневому екземплярі. З часом це стає некерованим. Натомість краще розбити код на модульні частини, щоб з ними було легше та гнучкіше працювати.

```
Vue.component('component-name',{
  template: '<div>This is a component view</div>',
  data(){ return {}},
  methods: { },
  computed: { }
})
```

Рис. 2.16 – Створення компоненту.

Тепер можна використовувати цей компонент як власний елемент всередині кореневого екземпляра Vue:

```
new Vue({ el: '#app' })
<div id="app">
    <component-name></component-name>
</div>
```

Рис. 2.17 – Використання компоненту.

Компоненти можна повторно використовувати скільки завгодно разів. Все через те, що компоненти також є екземплярами Vue, і кожного разу, коли використовується компонент, створюється новий його екземпляр. Параметр data компонента повинен бути функцією (а не об'єктом, як у кореневій інстанції), щоб кожен екземпляр міг підтримувати незалежну копію повернутого об'єкта даних.

Основна структура, яка зазвичай спостерігається при запуску програми з нуля, має папку з компонентами, а також main.js файл входу та App.vue компонент. У всіх реальних додатках також є загальні компоненти, такі як кнопки, панелі, модулі та інші компоненти інтерфейсу. Таким чином, можна створити іншу папку, яка називається ui всередині папки компонентів. Після

створення компонентів буде. можливість імпортувати будь-який з них куди завгодно і безпосередньо використовувати.

Оскільки додаток розділений на компоненти, то для того, щоб зробити їх багаторазовими, також потрібно передати деякі дані компонентам з батьківського компоненту. Для цього використовується props.

```
Vue.component('child-component', {
  props: ['someText'],
  template: `<div>{{ someText }}<div>`
});
new Vue({
  el: "#app",
  data() {
    return {
      textToPass: 'Hey! I am passed into child component'
    }
  }
});
<div id="app">
  <child-component someText="textToPass"></child-component>
</div>
```

Рис. 2.18 – Використання props.

В компоненті створено “someText” props і тому можна передавати дані з батьківського компоненту.

Що відбувається, коли є два компоненти з незначними варіаціями, зміст, або стильові відхилення? Можна передавати весь різний вміст і стилі вниз до компонента за допомогою props та перемикати все щоразу, або можна було б розділити самі компоненти та створити різні їх версії. Але було б дуже добре, якби була можливість повторно використовувати компоненти і заповнити їх тими ж даними або функціоналом. Ось тут слоти (<slot></slot>) дійсно зручні.

Щоб дозволити батьківському компоненту передавати елементи DOM в дочірній компонент, треба створити <slot></slot> елемент всередині дочірнього компонента. Потім можна буде заповнити елементи в цьому слоті за допомогою: <child-component> <h3>Текст</h3> </child-component>.

Іноді може знадобитись використання кількох слотів в одному компоненті. У цьому випадку можна використати іменовані слоти.

```
//base layout component
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
//any page
<base-layout>
  <template slot="header">
    <h1>Page title or smth. else</h1>
  </template>

  <p>Your main content here</p>

  <template slot="footer">
    <p>Footer related information here</p>
  </template>
</base-layout>
```

Рис. 2.19 – Використання іменованих слотів.

Якщо батьківський елемент не має жодного вмісту, то слот виведе все, що знаходиться в середині нього. Наприклад, в даному випадку буде виведено за замовчуванням Hello from the child!

```
<template>
  <div>
    <slot>
      <p>Hello from the child!</p>
    </slot>
  </div>
</template>
```

Рис. 2.20 – Приклад роботи slot.

2.4 Технологія Redux

Redux - одна з найпопулярніших бібліотек у розробці на сьогоднішній день. Однак часто плутаються з приводу того, що це таке і які його переваги. Як зазначено в документації, Redux є передбачуваним контейнером стану для додатків JavaScript . Іншими словами це архітектура потоку даних в додатках, а не традиційна бібліотека типу AngularJS. Redux використовується здебільшого для управління об’єктом state додатків. Підсумовуючи це, Redux підтримує стан цілої програми в єдиному непорушному дереві (об'єкті), яке неможливо змінити безпосередньо. Коли щось змінюється, створюється новий об’єкт (використовуючи дії та редуктори).

2.4.1 Переваги Redux

Переваги заключаються в [6]:

- передбачуваності результату. Завжди є одне джерело істини - store, який не має плутанини щодо того, як синхронізувати поточний стан з діями та іншими частинами програми.
- «ремонтопридатності». Завдяки передбачуваному результату та суворій структурі полегшує підтримку коду.
- організації. Redux змушує організувати код більш структуровано, що робить код більш послідовним та простішим для роботи команди.
- відображенні сервера. Це дуже корисно, особливо для початкової візуалізації, що сприяє кращій користувальницькій роботі або оптимізації пошукових систем. Просто треба передати створений на сервері store на сторону клієнта.
- інструментах для розробників. Розробники можуть відслідковувати все, що відбувається в додатку в режимі реального часу, від дій до змін стану.
- спільноті та екосистемі. Наявність спільноти в Redux робить її ще більш привабливою для використання.
- простоті тестування. Перше правило написання тестового коду - це написання невеликих функцій, які виконують лише щось одне і

незалежне. Код Rudux - це в основному функції, які є маленькими, зрозумілими та ізольованими.

2.4.2 Функціональне Програмування

- Redux був побудований на основі функціональних концепцій програмування:
- здатний складати функції як першокласні об'єкти.
- здатний передавати функції як аргументи.
- здатний керувати потоком за допомогою функцій, рекурсій та масивів.
- здатний використовувати чисті, рекурсивні функції вищого порядку, замикання та анонімні функції.
- здатний використовувати допоміжні функції, такі як map, filter and reduce.
- здатний поєднувати функції разом.
- state не змінюється (тобто він непорушний).
- порядок виконання коду не важливий.

Функціональне програмування дозволяє писати більш чистий і модульний код. Записуючи менші та простіші функції, які виокремлюються за обсягом та логікою, можна значно полегшити код для тестування, обслуговування та налагодження. Тепер ці менші функції стають кодом для багаторазового використання, і це дозволяє писати менше коду, що, звісно, є великою перевагою. Функції можна скопіювати та вставити будь-де без будь-яких змін. Функції, які є ізольованими за обсягом і виконують лише одне завдання, менше залежатимуть від інших модулів у додатку, і це зменшене з'єднання - ще одна перевага функціонального програмування.

Чисті функції повертають нове значення на основі переданих їм аргументів. Вони не змінюють існуючі об'єкти, натомість вони повертають нові. Ці функції не покладаються на стан, з якого вони викликані, і вони повернуть тей самий результат для будь-якого наданого аргументу. З цієї причини вони дуже передбачувані. Оскільки чисті функції не змінюють

жодних значень, вони не впливають ні на область застосування, ні на помітні побічні ефекти, а це означає, що розробник може зосереджуватися лише на значеннях, які повертає чиста функція.

2.4.3 Використання Redux

Більшість розробників пов'язують Redux з React, але його можна використовувати з будь-якою іншою бібліотекою. Наприклад, з Redux та AngularJS , Vue.js, Polymer, Ember, Backbone.js і Meteor. Але Redux разом із React є найпоширенішою комбінацією.

2.4.4 Будівельні частини Redux

Концепції Redux на вигляд складні або не зрозумілі, але на ділі вони досить прості. Цікаво, що бібліотека займає лише 2 Кб. Redux має три складові частини: дії, зберігання та редуктори (не враховуючи інтерфейс).

Коротше кажучи, дії - це події. Дії надсилають дані із програми в store. Store отримує інформацію лише від дій. Внутрішні дії - це прості об’єкти JavaScript, які мають властивість type (зазвичай константу), описуючи тип дії та корисну інформацію, що надсилається в store.

```
{
    type: LOGIN_FORM_SUBMIT,
    payload: {username: 'alex', password: '123456'}
}
```

Рис. 2.21 – Внутрішні дії.

Дії створюються разом зі створювачами дій. Вони є функціями, які повертають дії.

```
function authUser(form) {
  return {
    type: LOGIN_FORM_SUBMIT,
    payload: form
  }
}
```

Рис. 2.22 – Приклад функції.

Отже, викликати дії в будь-якій точці програми дуже легко, використовуючи такий dispatch метод:

```
dispatch(authUser(form));
```

Рис. 2.23 – Виклик dispatch методу.

Редуктор заснований на методі зменшення масиву, де він приймає зворотний виклик (редуктор) і дозволяє отримувати одне значення з декількох значень, сум цілих чисел або накопичення потоків значень. У Redux редуктори - це функції (чисті), які приймають поточний стан програми та дії і потім повертають новий стан. Дуже важливо розуміти, як працюють редуктори, оскільки вони виконують більшу частину роботи. Нижче буде наведений приклад дуже простого редуктора, який буде спочатку брати аргумент поточного стану та дії, а потім повертатиме наступний стан:

```
function handleAuth(state, action) {
  return _.assign({}, state, {
    auth: action.payload
  });
}
```

Рис. 2.24 – Приклад редуктора.

Для більш складних додатків можливе використання combineReducers() утиліти, наданої Redux. Вона поєднує всі редуктори в додатку в єдиний

редуктор індексів. Кожен редуктор відповідає за свою частину стану додатка, а параметр стану є різним для кожного редуктора. Утиліта combineReducers() робить структуру файлів набагато простішою в обслуговуванні. Якщо об'єкт (стан) змінює лише деякі значення, Redux створює новий об'єкт. Значення, які не змінювалися, стосуватимуться старого об'єкта і створюватимуться лише нові значення. Це дуже добре для продуктивності.

Store – це об'єкт, який містить стан програми та надає кілька допоміжних методів доступу до стану, диспетчерських дій та реєстрації слухачів. Весь state представлений одним store. Будь-яка дія повертає новий стан через редуктори.

```
import { createStore } from "redux";
let store = createStore(rootReducer);
let authInfo = {username: "alex", password: "123456"};
store.dispatch(authUser(authInfo));
```

Рис. 2.25 – Приклад роботи зі store.

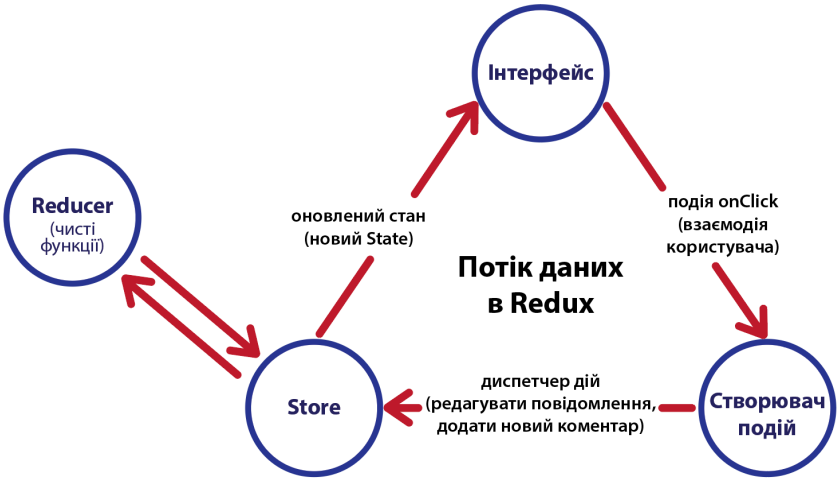


Рис. 2.26 – Потік даних в Redux.

2.5 Фреймворк Express.js

Експрес [7] - це швидкий, мінімалістичний веб-фреймворк для Node.js. Можна сказати, що це веб-додаток для Node.js. Він надає різні функції, які роблять розробку веб-додатків швидкою та легкою, що в іншому випадку вимагає більше часу, використовуючи лише Node.js. Express.js базується на модулі середнього програмного забезпечення Node.js, який

називається connect, який, у свою чергу, використовує модуль http . Отже, будь-яке проміжне програмне забезпечення, яке базується на підключенні, також буде працювати з Express.js.

Загалом фреймворк Node.js, Express.js - це структура, яка використовується для розробки веб-додатків на сервері. Він функціонує як легке надлаштування, яке працює поверх node.js. Завдяки надійному набору функцій, Express.js має неприхильний характер. Ця гнучкість означає, що її можна легко інтегрувати з проміжними програмами. Це означає, що ми можемо використовувати Express.js для швидкого створення складних та надійних додатків. Розробка веб-програми за допомогою Express.js дає змогу налаштувати додаток відповідно до потреб у межах його можливостей. Це означає, що ми можемо додавати або видаляти компоненти за необхідності, щоб переконатися, що ваша програма виконує саме те, що вам потрібно. Гнучкість Express.js полегшує розширення масштабів, коли бізнес цього потребує, а це означає, що у міру зростання бізнесу зростає і сама програма.

Переваги Express.js:

- Робить розробку веб-додатків Node.js швидкими та простими.
- Легко налаштувати.
- Дозволяє визначати маршрути програми на основі методів та URL-адрес HTTP.
- Включає різні модулі проміжного програмного забезпечення, які можна використовувати для виконання додаткових завдань на запит та відповідь.
- Легко інтегрується із різними двигунами шаблонів, такими як Jade, Vash, EJS тощо.
- Дозволяє визначити помилку при обробці середнього програмного забезпечення.
- Легко обслуговувати статичні файли та ресурси програми.
- Дозволяє створювати сервер API REST.
- Легко підключається до таких баз даних, як MongoDB, Redis, MySQL

Висновки до розділу 2

В даному розділі були описані популярні на сьогоднішній день фреймворки, а також методи та підходи до розробки сайтів чи веб додатків. Було коротко описано основні можливості та базові команди, які можуть виконувати ці бібліотеки, з наведенням прикладів у вигляді скріншотів коду. Після того, як були проаналізовані всі «за» та «проти», був зроблений висновок: побудувати проект за допомогою препроцесору Sass, фреймворків React.js, Redux та Express.js.

РОЗДІЛ 3

ОПИС ПРОГРАМНОГО ПРОДУКТУ ТА ЙОГО РЕАЛІЗАЦІЇ

3.1 Структура проекту

Оскільки весь проект був побудований на основі React, то всі файли розміщуються в папці my-app, котра включає в себе основу: папку node_modules, src, файли .gitignore, package.json та певний ряд папок із JavaScript файлами для роботи із фреймворком Express.js. В свою чергу node_modules включає в себе всі модулі, котрі використані в проекті. Версії всіх використаних бібліотек, модулів та фреймворків записані у файлі package.json. В папці src розміщена папка з компонентами React.js, App.js – головний компонент додатку та index.js, в котрому і створюється самий додаток.

Все це зроблено з метою швидкої та зручної взаємодії з системою керування версіями git. В цій роботі використовувався саме github.

Система керування версіями [8] – програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D-моделі, текстового документу тощо. Система керування версіями — інструмент, який дозволяє одночасно, не заважаючи один одному, проводити роботу над груповими проектами. Системи керування версіями зазвичай використовуються при розробці програмного забезпечення для відстеження, документування та контролю над поступовими змінами в електронних документах: у сирцевому коді застосунків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють декілька людей. Кожна версія позначається унікальною цифрою чи літерою, зміни документу занотовуються. Зазвичай також зберігаються дані про автора зробленої зміни та її час.

Також проект створювався за допомогою Node.js [9] – величезної платформи призначеної для виконання високопродуктивних

мережевих застосунків, написаних мовою програмування JavaScript. Платформа окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських та серверних програм. В платформі використовується розроблений компанією Google рушій V8. Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback). Для керування модулями використовується пакетний менеджер npm (node package manager).

Припустимо, що над проектом працювало б декілька чоловік. Для початку роботи над ним їм достатньо було б склонувати проект з гітхаб репозиторію за допомогою команди git clone, котру треба було б ввести в терміналі і після цього скористатися командою npm install, котра за допомогою пакетного менеджера npm встановить всі потрібні версії бібліотек, використані в даному проекті.

В папках із компонентами розміщені файли із самими компонентами та їх стилі, котрі написані за допомогою препроцесору Sass. Його можливості та переваги були описані в другому розділі. І останнє – файл .gitignore, котрий містить в собі назви папок та файлів, котрі не треба завантажувати на github репозиторій. Такими є node_modules. Немає необхідності її завантажувати на github репозиторій, оскільки версії всіх потрібних бібліотек записані в файлі package.json і при необхідності, за допомогою вище описаної команди, всі вони будуть встановлені в папку node_modules.

3.2 Використання React.js

Розглянемо використання React.js на прикладі створення головного меню. Меню складається із посилань та кнопок, тому необхідно створити компонент button.

```
import React from 'react';
import {Link} from "react-router-dom";
export default (props) => (
    <li className={props.className} id="sign_up" >
        <Link to={props.to}>{props.text}</Link>
    </li>
)
```

Рис. 3.1 – Створення компоненту button.

Для роботи з React.js необхідно його імпортувати. Також треба імпортувати Link, щоб зробити маршрутизацію для додатку – «рух» по додатку, котрий буде здійснений при кліку на посилання.

Об’єкт props містить в собі дані, котрі необхідні для створення компоненту, відображення тексту та задання стилів. Саме тому отримуємо ім’я класу та шлях, куди повинен перейти додаток при натисканні на цей елемент.

Далі створюємо сам компонент меню – header.

```
import React from 'react';
import classes from './Header.module.sass'
import Button from './button/Button'
import {
    BrowserRouter as Router,
    Switch,
    Route,
    Link
  } from "react-router-dom";
```

Рис. 3.2 – Імпортування в компоненті header.

Імпорт необхідних компонентів та застосунків для створення компоненту. При розробці досить часто бувають проблеми із «перебиванням стилів». Наприклад, якщо в одному компоненті є певний елемент із назвою класу container, і він має певні стилі, та існує другий компонент, в котрому є теж елемент із такою ж назвою класу, то при заданні стилів буде виникати конфлікт і стилі будуть застосовуватись не коректно. Саме тому в даному додатку використовуються модулі для запобігання таким ситуаціям. При їх

використанні до назви класу, котра буде використовуватися, додаватиметься певний хеш, котрий буде унікальним. Тож коли треба буде отримати якусь назву класу, то достатньо буде звернутися до об'єкту classes, котрий в собі і буде містити потрібний клас, який 100% буде унікальним, бо вкінці буде доданий хеш.

```
export default () => (
    <Router>
        <div>
            <header>
                <div className={classes["menu-top"]}>
                    <li className={classes["menu-top__logo"]}>
                        <Link to="/">Montakarlo_dev</Link>
                    </li>
                    <label htmlFor="toggle">&#9776;</label>
                </div>
                <input type="checkbox" id = 'toggle' style = {{display: 'none'}}/>
                <nav>
                    <ul className={classes["menu"]}>
                        <Link to = "/home" id="home">Home</Link>
                        <Link to = "/search" id="search">Search</Link>
                        <Link to = "/profile" id="profile">Profile</Link>
                        <li className={classes["menu__buttons"]}>
                            <ul className={classes["buttons-list"]}>
                                <Button
                                    className={classes["buttons-list__item_1"]}
                                    text = "Sign in"
                                    to = 'sign_in'
                                />
                                <Button
                                    className={classes["buttons-list__item_2"]}
                                    text = "Sign up"
                                    to = 'sign_up'
                                />
                            </ul>
                        </li>
                    </ul>
                </nav>
            </header>
        </div>
```

Рис. 3.3 – Створення компоненту header.

Все те, що буде включати в себе компонент, обгортається в компонент <Router> </Router>, оскільки необхідно налаштувати маршрутизацію додатку для всіх пунктів меню.

При створенні елементів отримуємо назви класів через classes, як було описано вижче.

Компонент Link використовується для реалізації маршрутизації. Шлях задаємо в параметрі to = “ ” .

Деякі пункти меню реалізовані, як посилання через компонент Link, котрий всередині і є звичним тегом <a>. Останні два пункти меню – це кнопки Sign in та Sing up для входу та реєстрацію. Для них був попередньо створений компонент, котрий тут і буде використаний. Із головного компоненту передаються потрібні параметри в компонент button за допомогою props. Оскільки обрані дві кнопки матимуть різні стилі, то їм задані різні назви класів. Також передаються такі параметри, як текст, котрий буде відображатися всередині та шлях, по котрому повинен перейти додаток при кліку на цей компонент.

```
            <Switch>
                <Route path="/home">
                    <Home />
                </Route>
                <Route path="/search">
                    <Search />
                </Route>
                <Route path="/profile">
                    <Profile />
                </Route>
            </Switch>
    </Router>
)
```

Рис. 3.4 – Реалізація вкладеної маршрутизації.

За допомогою <Switch/> <Switch/> буде визначатися, котрий саме компонент буде рендеритися при переході по певному маршруту. В середині нього в компонент <Route></Route> передається параметр path, котрий перевіряє шлях. Якщо поточний шлях співпадає із тим, що вказаний в параметрі path, то буде відображений той вміст, котрий знаходиться в середині компонента <Route> </Route>. В нього і поміщаємо саме ті компоненти, які

необхідно відобразити, а саме: домашня сторінка, сторінка з пошуком або профіль користувача.

```
import React from 'react';
import Header from './header/Header'

function App() {
  return (
    <div className="Project">
      <Header />
    </div>
  );
}

export default App;
```

Рис. 3.5 – Розміщення нового компоненту в головний компонент <App />.

Далі вставляємо створений <Header /> в основний компонент <App />.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Рис. 3.6 – Розміщення основного компоненту в створений додаток.

І останнє – це розміщення створеного компоненту <App /> в файл index.js. Таким чином і була організована вся логіка додатку, котра була продемонстрована вижче.

3.3 Використання Sass

Стилізацію за допомогою Sass розглянемо теж на прикладі меню.

```
header
    display: flex
    position: fixed
    justify-content: space-between
    align-self: center
    background-color: #222
    font-size: 20px
    z-index: 100
    width: 100%
    a
        color: white

.menu-top
    display: flex
    align-items: center
    padding: 0 20px
```

```
.menu
    padding: 5px 15px 5px 20px
    display: flex
    align-items: center
    &>a
        color: white
        margin: 0 8px
        transition: all .3s
        &:hover
            color: #1f5d9f
    &>li
        padding: 7px 5px

label
    display: none
    font-size: 28px
    color: white
```

Рис. 3.7 – Загальна стилізація блоку меню.

Для скорочення і швидшого написання коду використано &, котрий відноситься до батьківського селектору. Завдяки цьому і вкладеності не треба повторювати ту частину, що знаходиться на рівень вище. Також швидкості додає одна із основних можливостей даного препроцесору – відсутність фігурних дужок та крапки з комою. Достатньо лише дотримуватися табуляції (відступів) для того, щоб зберігати ієрархію і не заплутатись. Знак & також можна використовувати для псевдоселекторів. В даному випадку це :hover, котрий змінює або задає нові стилі при наведенні на вибраний елемент. Для швидкої та зручної стилізації використовуються flex-box – це основне, на що можна звернути увагу, все інше – досить базові та прості властивості.

```sass
$current_color: #2169aa
$hover_color: #1d5b89

.buttons-list
    display: flex
    &__item
        &_1
            display: flex
            align-items: center
            padding: 0 7px
            &>a
                height: 40px
                display: flex
                align-items: center
                justify-content: center
                width: 95px
                font-size: 18px
                white-space: nowrap
                background-color: $current_color
                border-radius: 20px
                transition: all .3s
                &:hover
                    background-color: $hover_color
                    cursor: pointer
        &_2
            @extend .buttons-list__item_1
            &>a
                background-color: transparent
                border: 1px white solid
                &:hover
                    background-color: #111111
                    cursor: pointer
```

Рис. 3.8 – Стилізація вмісту блоку меню.

В даній частині коду можна побачити більш цікаве використання можливостей Sass. Змінні $current_color та $hover_color створені для того, щоб потім можна було їх використовувати, просто звернувшись до них, а не шукати в коді саме цей колір, копіювати і потім вставляти. Знову ж таки використовується вкладеність, щоб повністю не переписувати назви класів. Для того, щоб користуватися цією можливістю на максимум, варто писати HTML (або в моєму випадку JSX) по технології БЕМ (Блок-елемент-модифікатор). Згідно з цією технологією іменування класів здійснюється послідовно додаючи до назви класу, котрий вижче по ієрархії, нове ім’я. Послідовність додавання «зашита» в самій назві технології: спочатку назва

зовнішнього блоку, потім – елементу в середині нього, далі – модифікатору для елементу.

```html
<div class="menu">
  <div class="menu__item">
    <div class="link"></div>
  </div>
  <div class="menu__item menu__item_current">
    <div class="link"></div>
  </div>
</div>
```

Рис. 3.8 – Приклад HTML коду за технологією БЕМ.

В коді, котрий зображено на рис. 3.8 також використовується директива @extend, за допомогою якої наслідуються стилі вибраного класу. Це дозволяє не переписувати ще раз всі стилі, котрі потрібні будуть для іншого класу. Якщо деякі з них треба замінити, то достатньо написати властивість з новим значенням і вона буде замінена.

```sass
@mixin border_style
    border-radius: 10px
    box-shadow: 0 0 4px rgba(0, 0, 0, 0.2)


.page-container__projects
    padding: 0 60px 0 500px
    display: flex
    flex-direction: row
    flex-wrap: wrap
    &-element
        overflow: hidden
        display: flex
        align-items: center
        justify-content: center
        margin: 0.7%
        width: calc(94.4% / 4)
        height: 13vw
        @include border_style
        &>img
            width: 100%
            height: 100%
            object-fit: cover
```

Рис. 3.9 – Приклад Sass коду з використанням міксину.

Міксини дозволяють записати стилі, котрі потім можуть бути використані повторно в якомусь іншому місці. В даному випадку, котрий зображений на рис. 3.9, використано міксин, оскільки далі буде багато класів, котрі будуть стилізовані однаково. Саме тому для прискорення роботи дуже зручно їх використовувати, щоб не шукати те, що вже десь було написано або не копіювати і заново вставляти одне і те ж саме.

3.4 Використання Express.js

Реєстрація користувача, отримання даних, редагування, додавання та видалення здійснюється за допомогою фреймворку Express.js. Структура серверних файлів організована таким чином: файл із функціями, котрі будуть необхідні для певних перевірок, обчислень або інших дій, головний файл, в котрому описані всі шляхи, файл, в котрому реалізовані всі запити на сервер, файл із валідацією даних, файл, котрий буде відповідати за повернення відповіді та файл, котрий є моделлю і включає в себе всі необхідні поля, котрі повинні бути заповнені. Є сенс описувати реалізацію запитів, валідацію та відповіді сервера, тож почнемо із запитів.

```
const { Router } = require('express');
const UserService = require('../services/userService');
const { responseMiddleware }
= require('../middlewares/response.middleware');
const { createUserValid, updateUserValid, deleteUserById}
= require('../middlewares/user.validation.middleware');

const router = Router();

router.get("/:id", (req, res) =>{
  let id = req.params["id"];
  if (UserService.search({'id': id})){
    res.status(200);
    res.send(UserService.search({'id': id}));

  } else {
    res.status(404);
    res.json({
      error: true,
      message: 'User with this id is not found'})
  }
})
```

Рис. 3.10 – Get запит.

Імпортовано Express.js для роботи із сервером та всі інші допоміжні файли, котрі будуть далі використані. Створено Get запит, котрий буде отримувати об’єкт із даними користувача. Функція приймає два параметри: request і response. В другому параметрі знаходиться відповідь від сервера, а в першому – всі параметри, котрі будуть отримані під час цього запиту (вони знаходяться в url). За допомогою функції search() здійснено пошук по id користувача. У відповіді сервер поверне статус 200, що означає успішність, і об’єкт із даними користувача із потрібним id. У випадку, якщо такого користувача не буде знайдено, сервер поверне помилку 404 із повідомленням про відсутність даних за таким id.

```
router.post("/", createUserValid, responseMiddleware, (req,res) => {
  UserService.addUser(req.body);
  let obj = req.body
  res.status(200);
  res.json(obj);
});
```

Рис. 3.11 – Post запит.

В Post запиті використовується вже більш складний підхід, котрий здійснюється за допомогою middlewares – функцій проміжної обробки, котрі мають доступ до відповіді серверу. З їх допомогою здійснюється перевірка введених даних та ін. Після цього запиту дані відразу перевіряються та обробляються за допомогою функції createUserValid, далі вони передаються до responseMiddleware і вже після цього буде отримана відповідь.

В createUserValid реалізована перевірка даних, котрі будуть вводитися в полях форми реєстрації. У випадку, якщо хоча б одна з умов не виконується, то далі в request записується код помилки та повідомлення і переходить до функції responseMiddleware. З неї повертається відповідь сервера у вигляді помилки та повідомлення про те, в якому саме вона місці. Якщо ж всі перевірки проходять, то програма переходить знову ж таки до функції responseMiddleware, а далі повертається до самого запиту Post і виконується код в середині нього. В такому випадку, коли виконання програми доходить

до запиту Post, сервер повертає статус 200 і об'єкт із зареєстрованим користувачем. Нижче буде представлений скріншот реалізації перевірки даних. Для зручності, читабельності та швидкості реалізації коду програми були створені функції тих перевірок, котрі будуть повторюватися в різних запитах.

```
const createUserValid = (req, res, next) => {
    let inputObj = req.body;
    let userKeys = Object.keys(user);
    userKeys = deleteFromArr(userKeys, 'id');
    let inputObjKeys = Object.keys(inputObj);
    let email = inputObj.email;
    let phoneNumber = inputObj.phoneNumber;
    let base = UserService.allUsers();

    if (!Object.keys(inputObj).length){
        req.body = [400, 'Request with empty data'];
        next();
    } else if (inputObjKeys.includes('id')){
        req.body = [400, 'Request does not have to exist an id field'];
        next();
    } else if (!ValidateFields(inputObjKeys, userKeys)){
        req.body = [400, 'Missing some fields'];
        next();
    } else if (email.slice(-10) != '@gmail.com'){
        req.body = [400, 'Incorrect email'];
        next();
    } else if (phoneNumber.slice(0,4) != '+380' || phoneNumber.length != 13){
        req.body = [400, 'Incorrect phone'];
        next();
    } else if (checkForSameKeyValue(inputObj, 'email', base)){
        req.body = [400, 'User with the same email already exist'];
        next();
    }else if (checkForSameKeyValue(inputObj, 'phoneNumber', base)){
        req.body = [400, 'User with the same phone number already exist'];
        next();
    } else if (inputObj['password'].length<3){
        req.body = [400, 'Incorrect password. Enter more than 3 symbols'];
        next();
    } else {
        req.body = deleteExternalFields(inputObj, user)
        next();
    }
}
```

Рис. 3.12 – Валідація даних при реєстрації нового користувача.

```
router.put("/:id", updateUserValid, responseMiddleware, (req,res) => {
  UserService.updateUser(req.params["id"], req.body);
  let obj = req.body
  res.status(200);
  res.json(obj);
});
```

Рис. 3.13 – Put запит.

В Put запиті такий самий підхід із використанням middlewares. Після цього запиту дані відразу перевіряються та обробляються за допомогою функції updateUserValid і передаються до responseMiddleware.

```
const updateUserValid = (req, res, next) => {
    let id = req.params["id"];
    let inputObj = req.body;
    let userKeys = Object.keys(user);
    userKeys = deleteFromArr(userKeys, 'id');
    let inputObjKeys = Object.keys(inputObj);
    let email = inputObj.email;
    let phoneNumber = inputObj.phoneNumber;
    let base = UserService.allUsers();
    if (!Object.keys(inputObj).length){
        req.body = [400, 'Request with empty data'];
        next();
    } else if (inputObjKeys.includes('id')){
        req.body = [400, 'Request does not have to exist an id field'];
        next();
    } else if (!ValidateFields(inputObjKeys, userKeys)){
        req.body = [400, 'Missing some fields'];
        next();
    } else if (email.slice(-10) != '@gmail.com'){
        req.body = [400, 'Incorrect email'];
        next();
    } else if (phoneNumber.slice(0,4) != '+380' || phoneNumber.length != 13){
        req.body = [400, 'Incorrect phone'];
        next();
    } else if (checkForSameKeyValue(inputObj, 'email', base)){
        req.body = [400, 'User with the same email already exist'];
        next();
    }else if (checkForSameKeyValue(inputObj, 'phoneNumber', base)){
        req.body = [400, 'User with the same phone number already exist'];
        next();
    }else if (inputObj['password'].length<3){
        req.body = [400, 'Incorrect password. Enter more than 3 symbols'];
        next();
    } else if (!checkForSameId("id", id, base)){
        req.body = [404, 'User with this id is not found'];
        next();
    } else {
        req.body = deleteExternalFields(inputObj, user)
        next();
    }
}
```

Рис. 3.14 – Валідація при оновленні даних користувача.

```
router.delete("/:id", deleteUserById, responseMiddleware, (req,res) => {
  UserService.deleteUser(req.params["id"]);
  let obj = req.body
  res.status(200);
  res.json(obj);
});
```

Рис. 3.15 – Delete запит.

В Delete запиті використаний такий самий підхід, як і в попередніх випадках . Логіка роботи та руху по коду програми теж ідентична.

```
const deleteUserById = (req, res, next) => {
    let id = req.params["id"];
    let base = UserService.allUsers();

    if (!checkForSameId("id", id, base)){
        req.body = [404, 'User with this id is not found'];
        next();
    } else {
        let a = getObjectByKey("id", req.params["id"], base)
        console.log("deleteUserById -> a", a)
        req.body = getObjectByKey("id", req.params["id"], base)
        next();
    }
}
```

Рис. 3.16 – Delete запит.

Для видалення користувача із бази даних треба спочатку перевірити чи існує взагалі користувач із таким id. Якщо так, то запит видалить цього користувача і поверне об’єкт з його даними, якщо ж ні – помилку із кодом 404 і повідомленням.

Нижче буде представлено функцію responseMiddleware, котра слугує так званим «провідником» для отримання відповіді від сервера. В будь-якому випадку відповідь буде отримана за допомогою цієї функції. Якщо запит помилковий, то код помилки буде порівняний і в response повернеться код помилки з повідомленням, необхідним для розуміння того, в чому є проблема при введені, редагуванні чи видаленні даних.

```
const responseMiddleware = (req, res, next) => {
    let obj = req.body

    if (obj[0] == 404){
        res.status(obj[0]);
        res.json({
        error: true,
        message: obj[1]})
    } else if(obj[0] == 400){
        res.status(obj[0]);
        res.json({
          error: true,
          message: obj[1]})
    } else {
        next()
    };
}
exports.responseMiddleware = responseMiddleware;
```

Рис. 3.17 – Реалізація функції responseMiddleware.

Висновки до розділу 3

В даному розділі було описано всі технології, фреймворки та бібліотеки, котрі використовувалися при розробці проекту. Була продемонстрована реалізація кожного етапу та кожної частини, на котрі можна умовно поділити створення продукту. На скріншотах зображені ділянки програми, котрі слугують прикладом підходів, які були обрані для вирішення конкретних задач (кожен із скріншотів був описаний із розгорнутим поясненням).

РОЗДІЛ 4

ВІЗУАЛЬНА СКЛАДОВА ПРОГРАМНОГО ПРОДУКТУ

4.1 Адаптація під усі види пристроїв

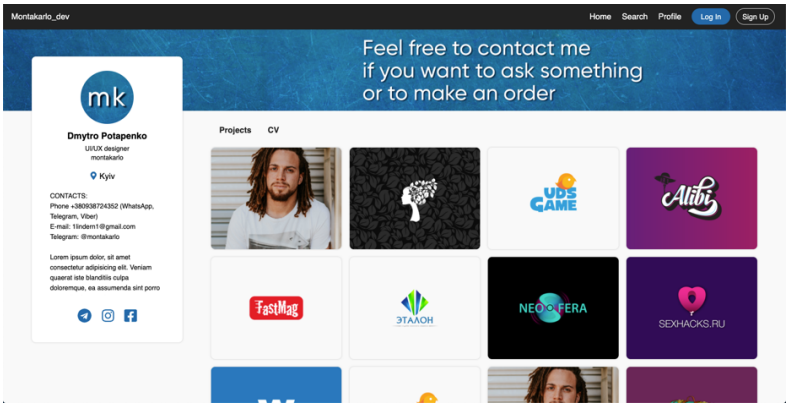


Рис. 4.1 – Вигляд для екранної роздільної здатності більше 1720px.

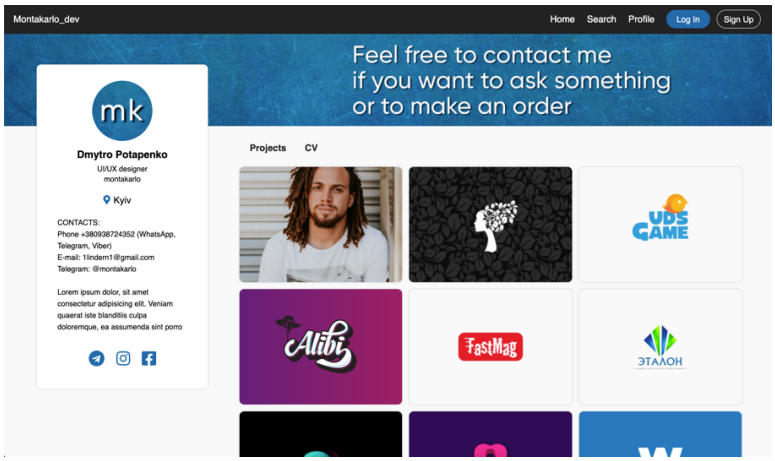


Рис. 4.2 – Вигляд для екранної роздільної здатності від 1489px до 1720px.

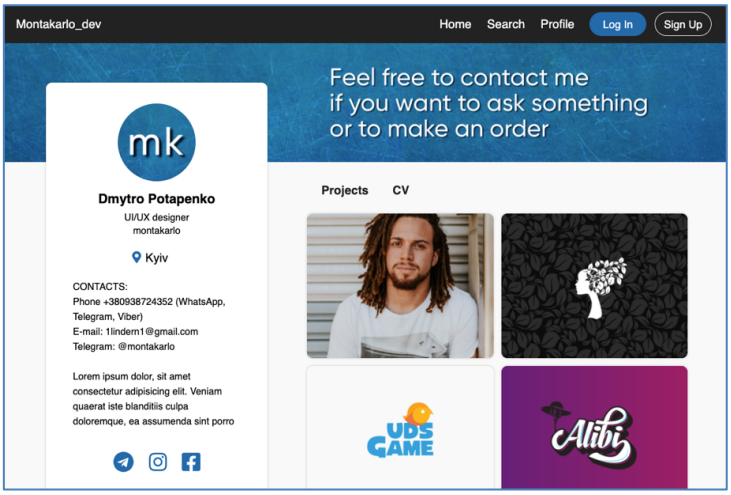


Рис. 4.3 – Вигляд для екранної роздільної здатності від 1200px до 1489px.

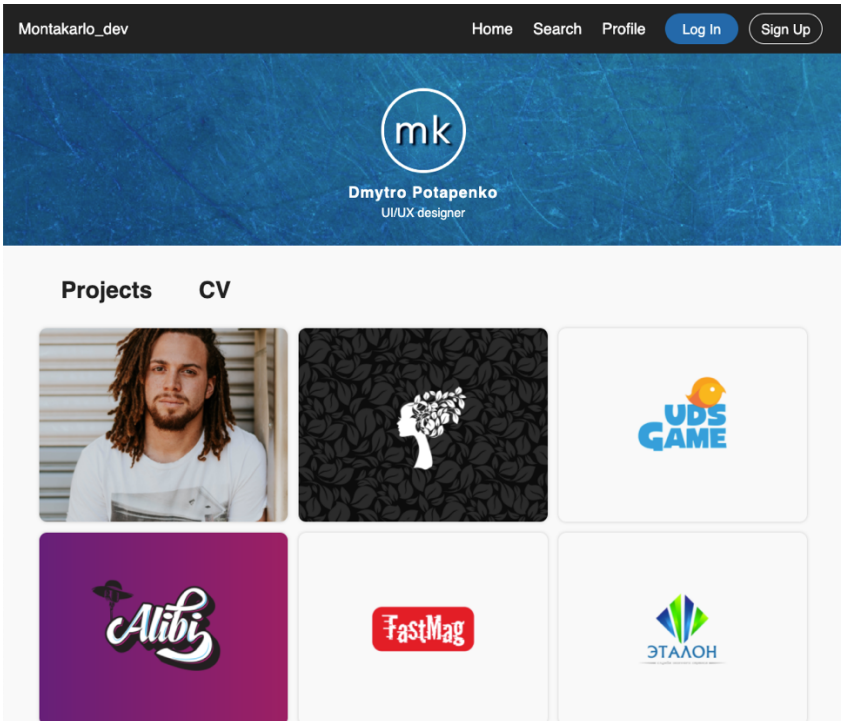


Рис. 4.4 – Вигляд для великих планшетів.

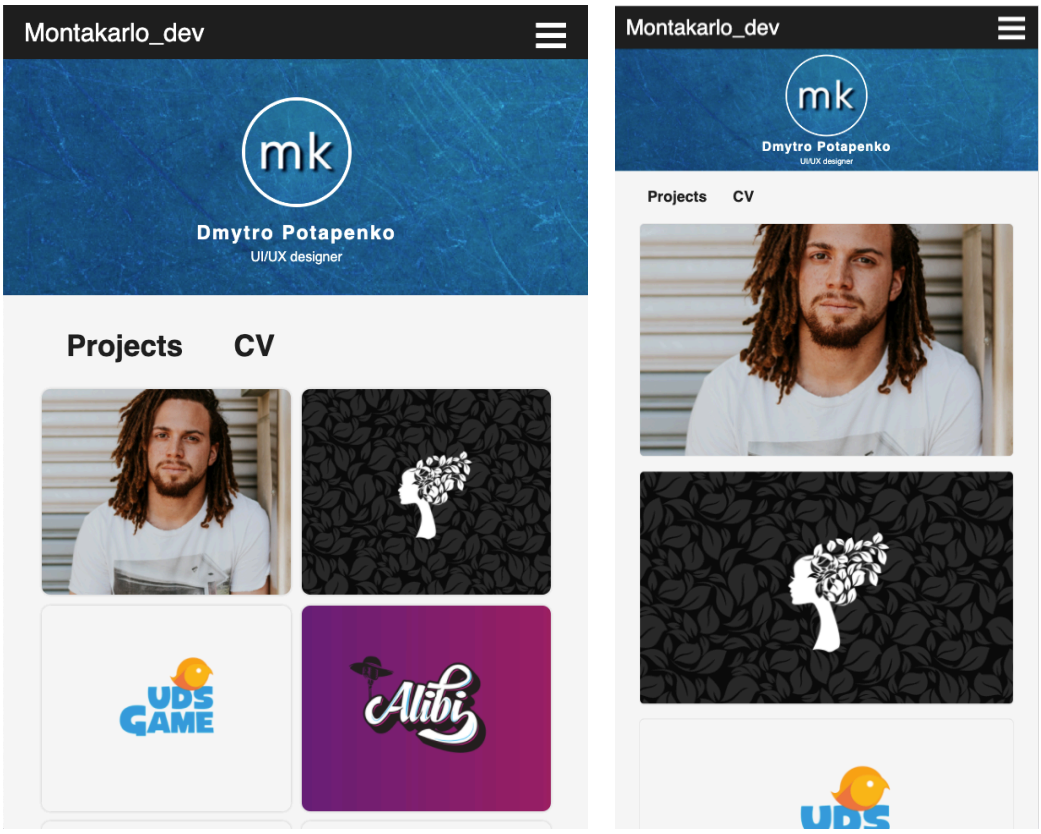


Рис. 4.5 – Вигляд для маленьких планшетів та мобільних пристроїв.

Для зручного користування сайтом на всіх типах пристроїв була реалізована його адаптація.

4.2 Наповнення сайту та його можливості

Окрім головної сторінки із прикладами робіт, котрі можна побачити вижче, є вкладка із резюме, де можна знайти всю необхідну інформацію.

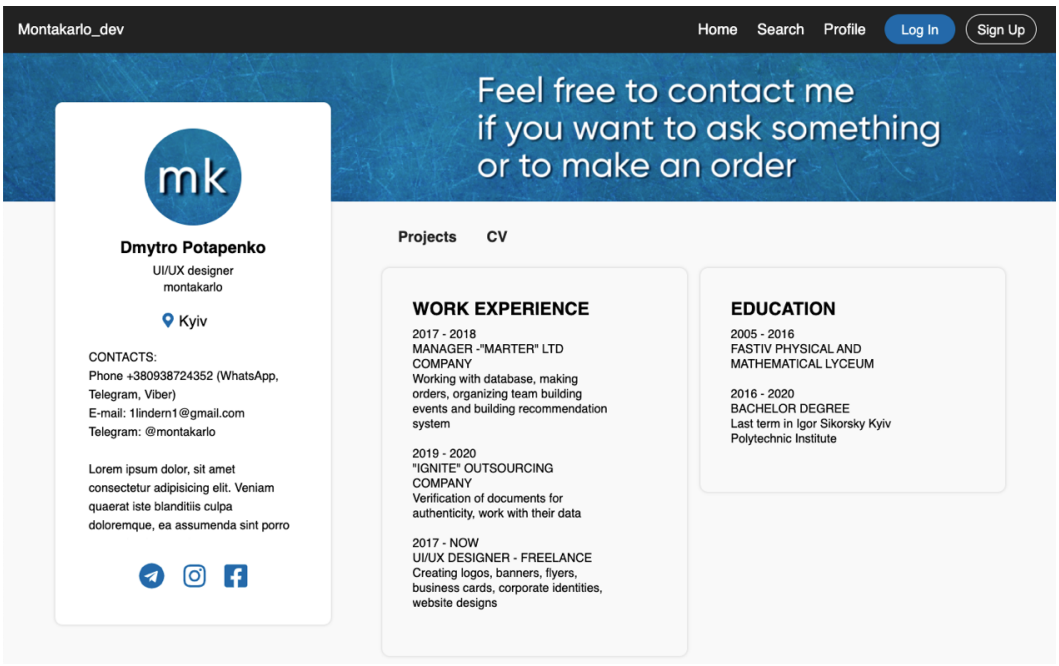


Рис. 4.6 – Вкладка із резюме.

Також користувачі мають можливість редагувати свій профіль в пункті меню Profile.

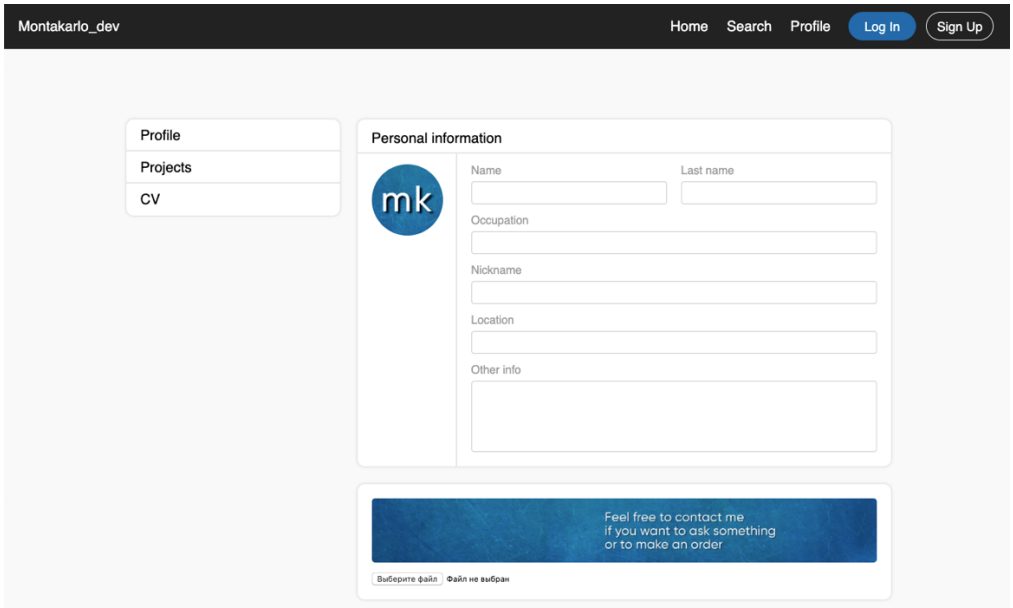


Рис. 4.7 – Налаштування профілю користувача: зміна особистих даних.

При переході в налаштування профіля користувач має можливість редагувати свої особисті дані, завантажувати нові роботи, або змінювати наповнення свого резюме.

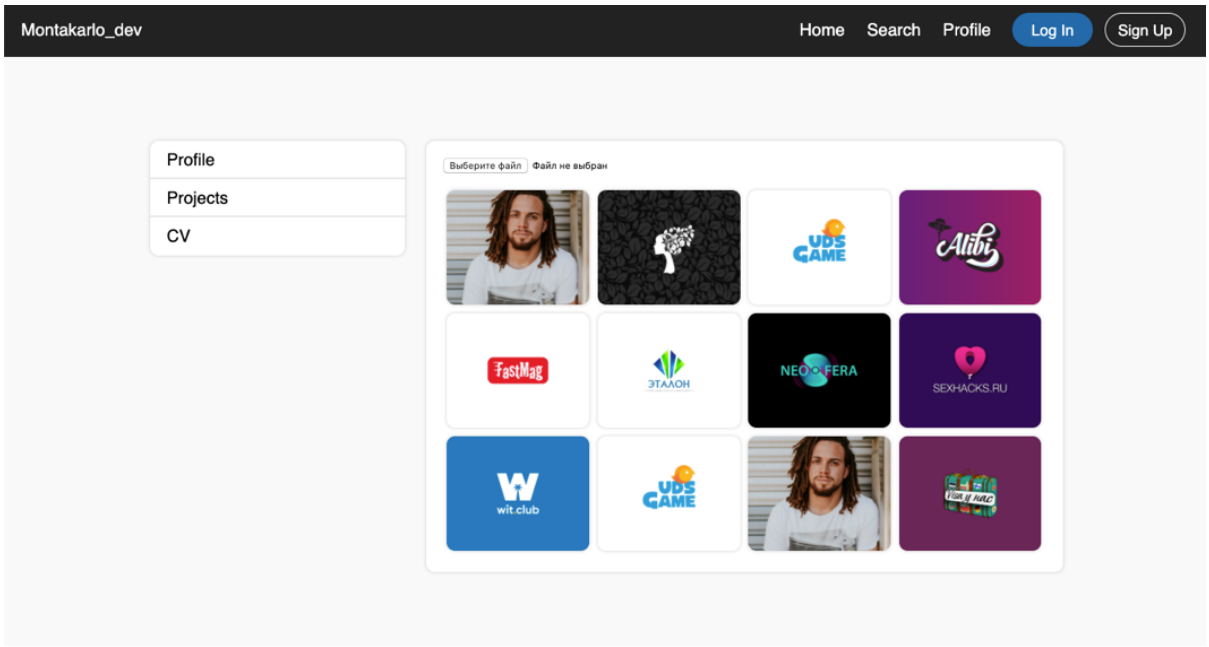


Рис. 4.8 – Налаштування профілю користувача: заміна робіт користувача.

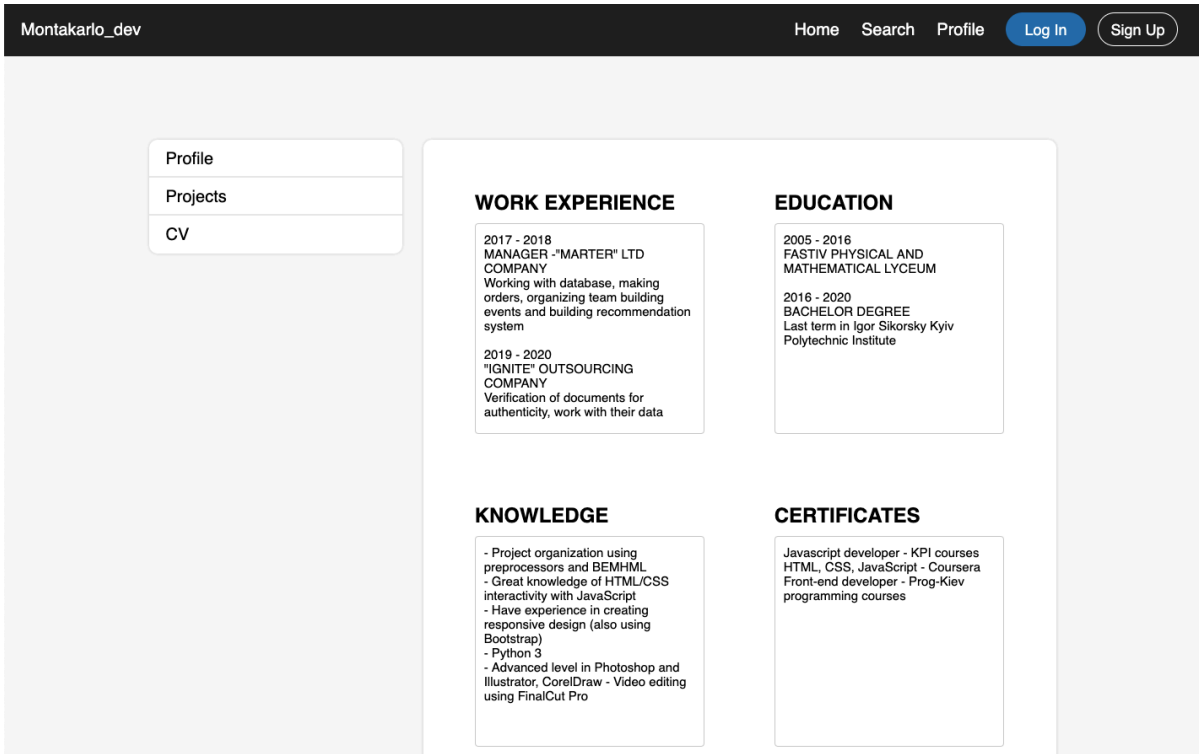


Рис. 4.9 – Налаштування профілю користувача: редагування резюме користувача.

Також, звісно користувач може зареєструватися і війти в свій акаунт. Для входу треба клікнути на кнопку Log in в меню.

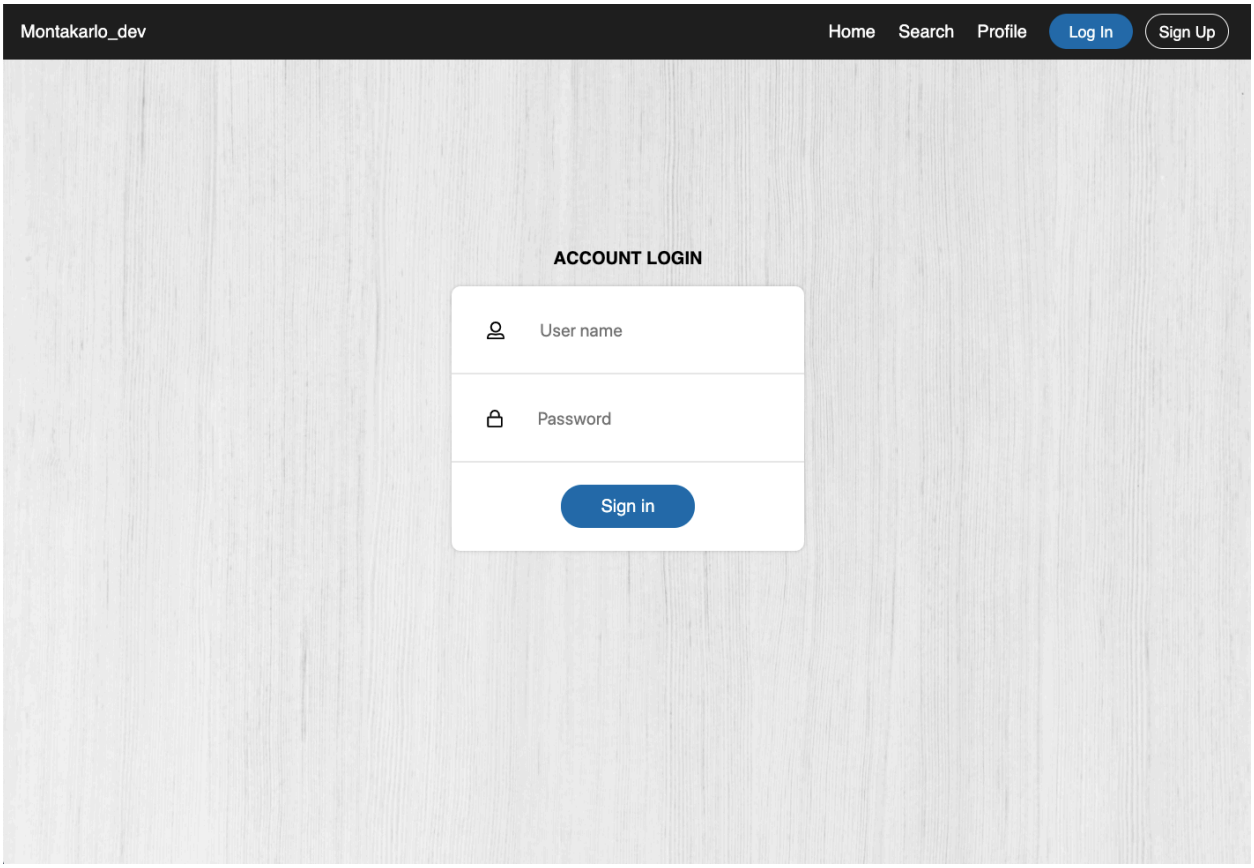


Рис. 4.10 – Форма входу користувача.

Висновки до розділу 4

В даному розділі було описано всі можливості та показано візуальну складову сайту. Користувачі можуть реєструвати профілі, заповнювати свої дані, завантажувати роботи і додавати резюме із необхідною інформацією. Дизайн сайту був зроблений максимально простим, зрозумілим та візуально приємним. Також він адаптований для різних пристроїв, щоб хто завгодно і при будь-яких умовах міг ним скористатися.

ВИСНОВКИ

Під час виконання даного дипломного проекту був розроблений сайт, котрий орієнтований на людей із творчими професіями. В загальному він дає можливість розміщувати свої роботи та резюме. Саме тому сервіс став не лише місцем, де можна виставити портфоліо, але й місцем, за допомогою якого можна знайти роботу або необхідну людину, котра потрібна для виконання певних задач чи найму в компанію тощо. Реалізація була здійснена за допомогою мови програмування JavaScript, його допоміжних бібліотек Node.js, React, jQuery, Express та допоміжного засобу – препроцесору Sass.

В першому розділі було описано швидкий розвиток веб технологій, котрі забезпечують комфортне використання Інтернету. Були розглянуті основні та базові технології, котрі дуже швидко змінювалися та модернізувалися з поширенням і популяризацією світової мережі. Також було описано альтернативу створення веб сторінок – конструктори сайтів, їх переваги та недоліки.

В другому розділі були описані популярні та актуальні фреймворки, а також методи та підходи до розробки сайтів чи веб додатків. Було коротко описано основні можливості та базові команди, які можуть виконувати бібліотеки. Найбільш поширені з них були використанні для створення проекту.

В третьому розділі були описані всі технології, фреймворки та бібліотеки, котрі використовувалися при розробці. Була продемонстрована реалізація кожного етапу та кожної частини, на котрі можна умовно поділити створення продукту. Кожен крок здійснювався поступово і вже вкінці всі розроблені частини були поєднані між собою, що створило готовий для використання продукт. Спочатку була створена візуальна складова сайту на основі JavaScript та його бібліотеки Redux, потім – стилізація всіх компонентів та адаптація під різні типи пристроїв за допомогою препроцесору Sass, котрий став швидкою та зручною альтернативою класичного Css, далі – серверна

частина так званого додатку, яка побудована на фреймворку Express.js. В поєднанні це і дало результат, котрий можна побачити в останньому розділі дипломної роботи.

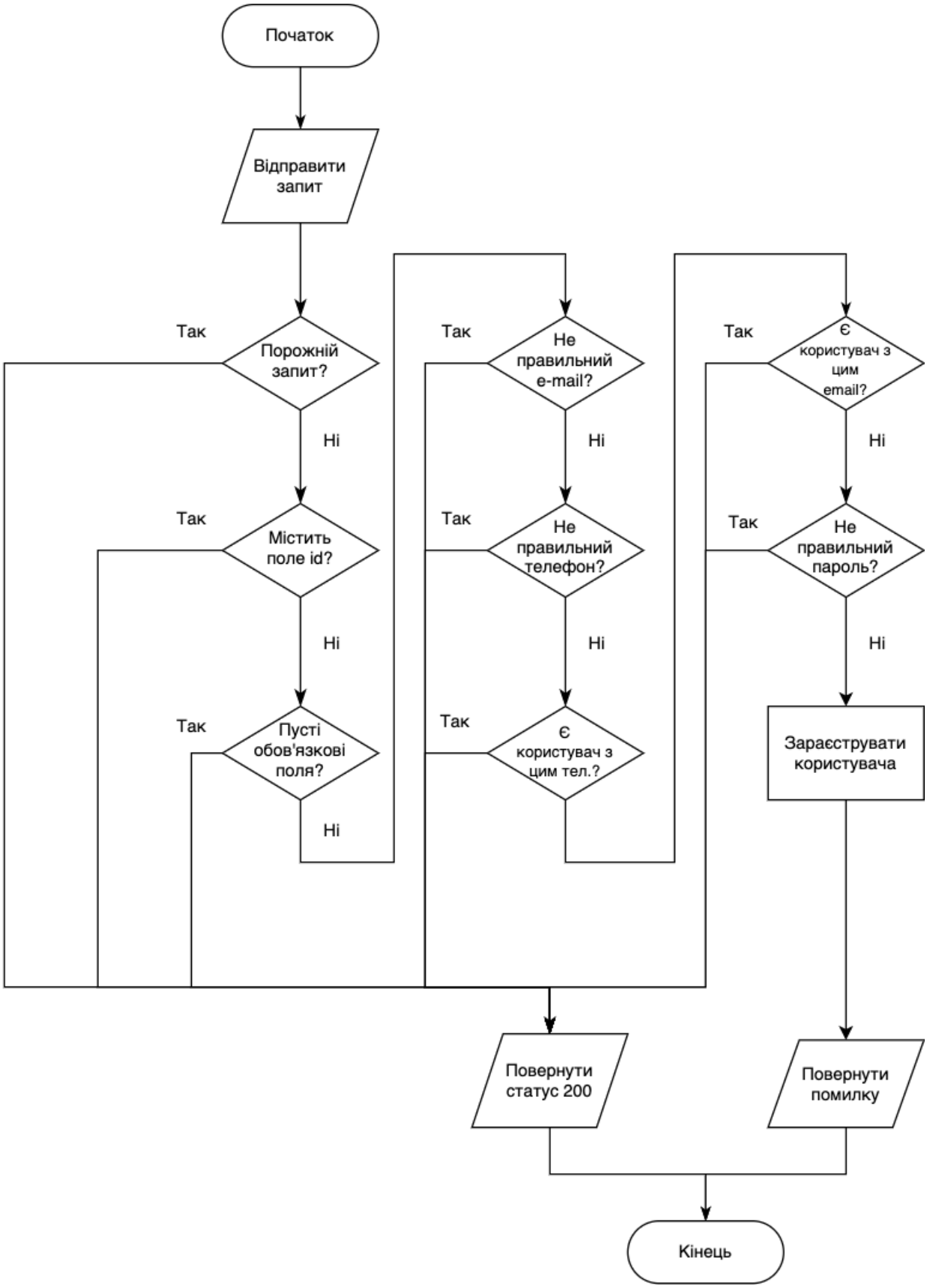
В четвертому розділі було описано всі можливості та показано візуальну складову сайту. Дизайн був зроблений максимально простим, зрозумілим та візуально приємним, без зайвих та незрозумілих вікон чи кнопок. Він був також адаптований для різних пристроїв для збільшення кола користувачів.

Створений сервіс не перевантажений різним функціоналом, котрий може бути незрозумілим для багатьох людей і є зручним для кожного користувача: як для того, яка шукає роботу, так і для роботодавця. Поєднання портфоліо і резюме дає можливість запровадити тренд швидкої відправки своїх основних даних лише одним посиланням – саме це і є новизною проекту. Зручний, мінімалістичний, інтуїтивно зрозумілий дизайн та адаптація під усі типи пристроїв пришвидшує та допомагає в заохоченні аудиторії.

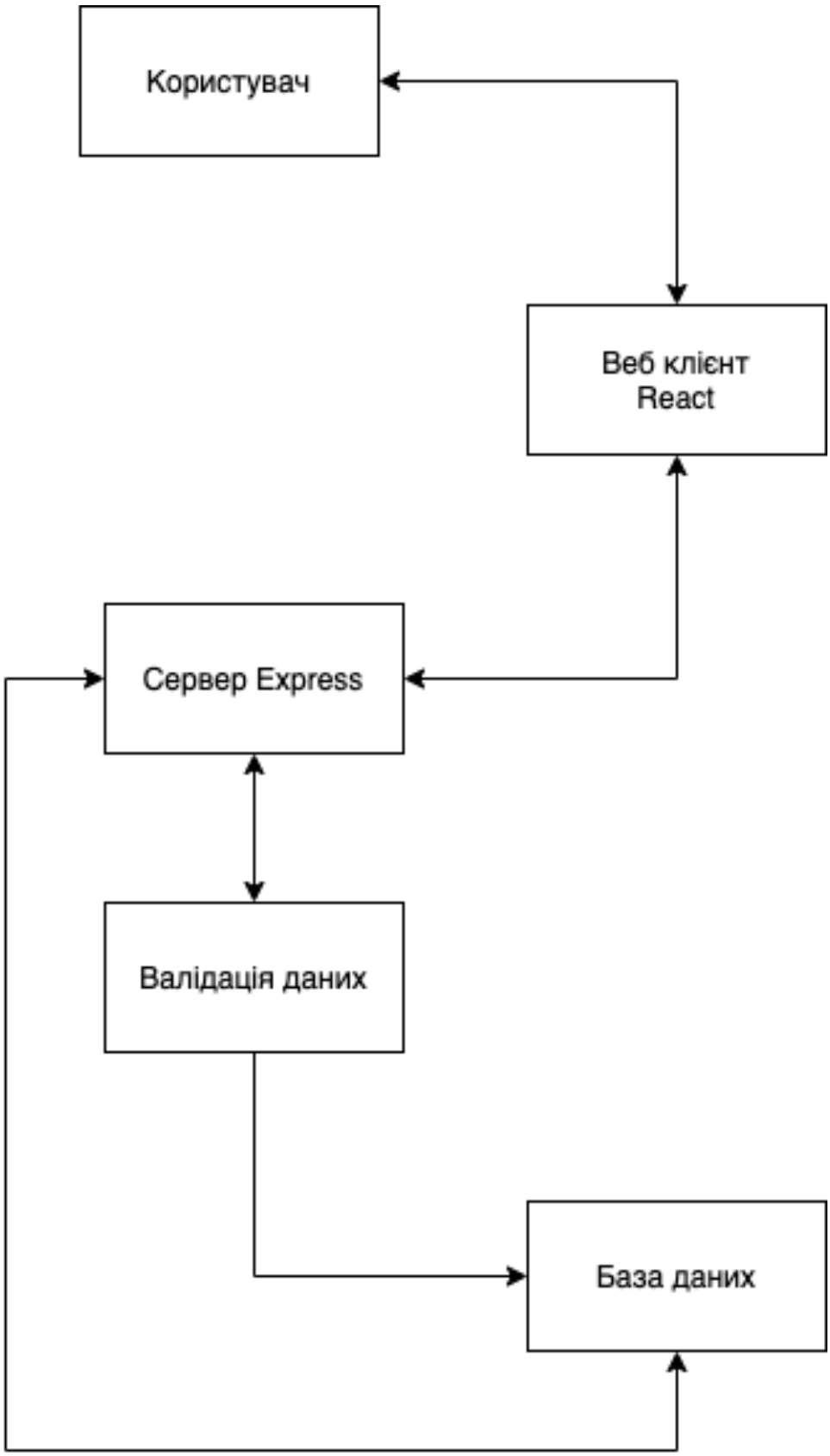
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brief History of the Internet [Електронний ресурс] – Режим доступу до ресурсу: https://www.internetsociety.org/internet/history-internet/brief-history-internet/.
2. Конструктор сайтов [Електронний ресурс] – Режим доступу до ресурсу: https://www.wix.com/.
3. Що таке Virtual DOM? [Електронний ресурс] – Режим доступу до ресурсу: https://codeguida.com/post/1561.
4. What about Angular? Angular JS history through the years [Електронний ресурс] – Режим доступу до ресурсу: https://www.ryadel.com/en/angular-angularjs-history-through-years-2009-2019/.
5. React + Redux: Architecture Overview [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/mofed/react-redux-architecture-overview-7b3e52004b6e.
6. ReactJS – JavaScript-библиотека [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/hub/reactjs/.
7. Что такое Express.js? – 8 Ответов [Електронний ресурс] – Режим доступу до ресурсу: https://overcoder.net/q/7215/%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-expressjs.
8. Система керування версіями [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D1%96%D1%8F%D0%BC%D0%B8.
9. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js#%D0%9E%D0%B3%D0%BB%D1%8F%D0%B4.

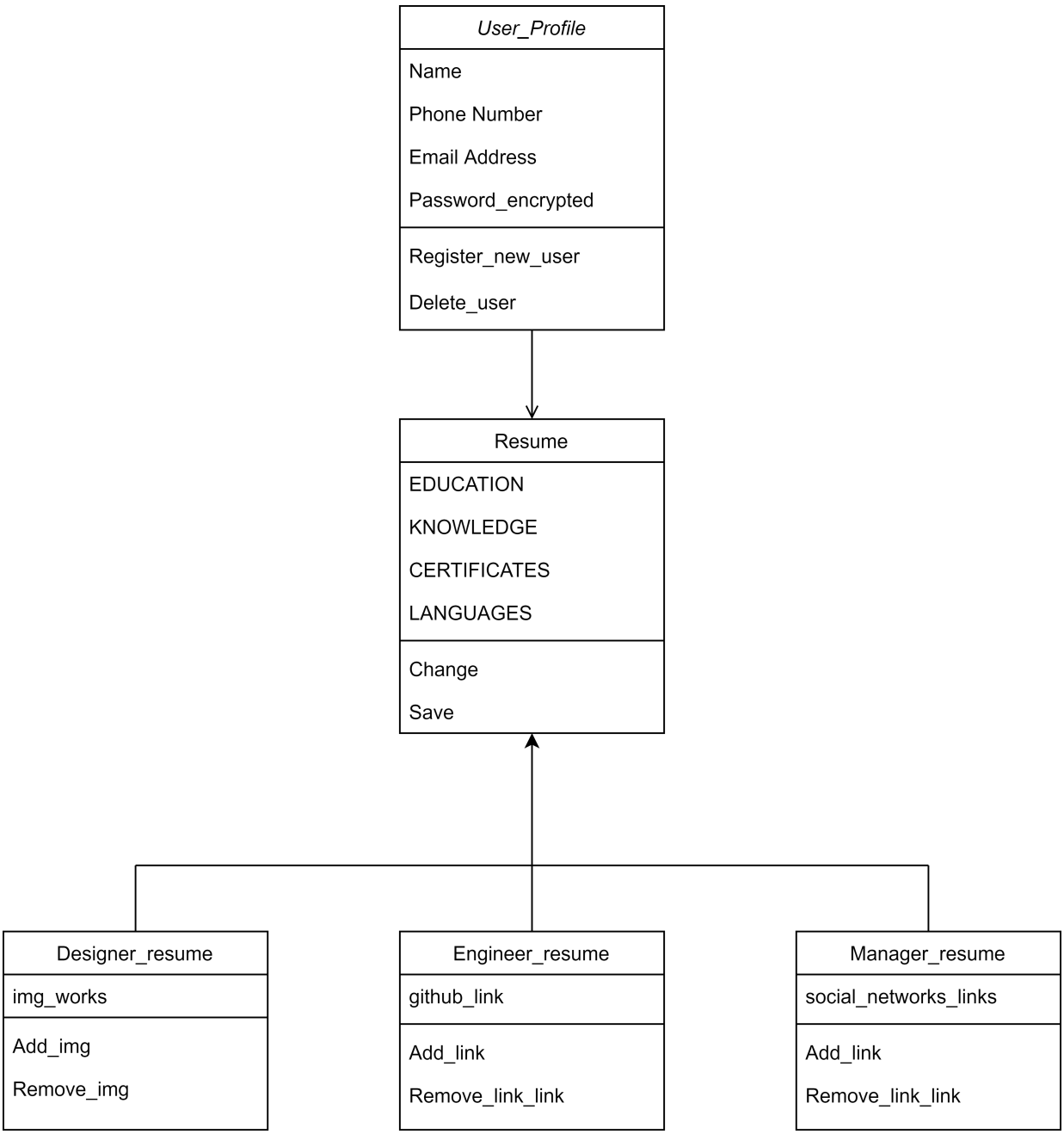
ДОДАТКИ



Зм.	Арк.	№ документа	Підп.	Дата	ДП.6421.04.000 Д1		
					Серверна система управління контентом Додаток А	Літ. Т	Аркуш 1 / Аркушів 1
Н.контр.						НТУУ «КПІ імені Ігоря Сікорського», ФІОТ Група ІО - 64	
Затв.		Симоненко В.П.					



					ДП.6421.05.000 Д2	
Зм.		№ документа	Підп.	Дата		
					Серверна система управління контентом Додаток Б	Літ. Т / Аркуш 1 / 1
Н.конатр		Симоненко В.П.				НТУУ «КПІ імені Ігоря Сікорського», ФІОТ Група ІО - 64
Затв.						



					ДП.6421.06.000 ДЗ		
Зм.		№ документа	Підп.	Дата			
					Серверна система управління контентом Додаток В	Літ.	Аркуш
						Т	1
Н.контр		Симоненко В.П.				НТУУ «КПІ імені Ігоря Сікорського», ФІОТ Група ІО - 64	
Затв.							